Solutions

ALM for Microsoft Dynamics CRM 2011: CRM Solution Lifecycle Management

MCS / OneTAP White Paper

VERSION 1.0

AUTHORS: Phil Hand, Balázs Töreki, Jürgen Leitz
COMPANY: Microsoft Corporation

CONTRIBUTORS: Roger Gilchrist

EXTERNAL RELEASE DATE: May 2013

Microsoft Dynamics CRM

Acknowledgements

This document is the result of an effort by Microsoft Consulting Services through the OneTAP Program, which is aligned with the Product group TAP/RDP programs. When a product is launched, Microsoft has the opportunity to demonstrate for customers a “One Microsoft” execution from product development through testing, deployment, and training, which helps to ensure the best customer experience possible. To further this goal, Microsoft relies on a functional enterprise readiness ecosystem that leverages and consolidates the guidelines and best practices that are derived from these customer engagements and reproduces them in a consistent and timely manner. Through complete execution of the OneTAP program, Services works with the Product Groups to achieve product signoff for enterprise Readiness.

Table of Contents

Business Summary

As Microsoft Dynamics CRM continues pushing towards the enterprise space, it becomes increasingly important to support the requirements of the enterprise—not only through the product but also in terms of functionality, scale, resilience, and security. From an implementation perspective, enterprises expect to be able to have structured, repeatable processes that are predictable and well documented to manage their application lifecycle. To date, a number of approaches have been articulated through various channels and from a variety of sources, but a definitive and full lifecycle approach has not been concisely communicated by Microsoft. Many customers, partners, and consultants within MCS are reaching out for best practice advice and guidance, and it is Microsoft's responsibility to deliver on this expectation. This document focuses on providing a coherent end-to-end approach to enterprise development for Dynamics CRM.

Introduction

A robust approach to application lifecycle management is at the heart of all good software application development; applications built upon the Dynamics CRM platform are no exception to this rule. The advent of Solution Packages has improved the capability to produce and control how components of a Dynamics CRM application interact with each other, thus enabling independent software vendors (ISVs) and IT Departments to build upon their functionality and to take advantage of component reuse.

Note: A table listing of the acronyms and associated terms used in this paper is provided in Appendix A: Acronyms.

At the core of any software application, there must be a well-defined, structured, and coherent approach to version control, build, and release management. This document discusses these topics and how they apply in a Dynamics CRM context.

The overview sections in this introductory chapter provide high-level insights into the content that will be more closely examined in the remainder of the document.

Methodologies Overview

The existing Microsoft solution delivery methodologies already provide coverage for enterprise application delivery. These methodologies are:

- **Dynamics Sure Step**: Microsoft's full customer lifecycle methodology supports a product-specific delivery approach for all Microsoft Dynamics solutions.
- **Microsoft Solution Framework**: Microsoft's general methodology and framework provides important, flexible principles and a mindset to achieve end-to-end quality and consistency across phases, projects, and teams on any delivery – balancing between waterfall and iterative approaches.
- **Visual Studio Team System/TFS**: This methodology, an entire application lifecycle management (ALM) environment with guidance and toolset, provides fundamental services such as version control, work item and bug tracking, build automation, test management, data warehouse, and a fully customizable and extendable framework to be able to support any delivery processes.

Note: Additional information about existing methodologies is provided in Appendix B: Methodologies.

Enterprise Project Delivery Overview

When implementing a business application on the Dynamics CRM platform, a number of key topics must be considered. These topics are similar to those that would apply for any enterprise software application:

- Project length, number of iterations
- Affected organization size, user base
- Multi-site, multi-language, multi-tenancy

- Requirement level, fit-gap complexity
- Integration level and type, number of connected systems
- Solution architecture complexity, position of Dynamics CRM in architecture
- Existing functionality and data(upgrade/migration)
- Reusability (generating future IP, reusing existing IP)
- Development team and phase complexity (parallel teams, customer-supplier teams, IT-business teams)
- Project structure, management, delivery methodology, risk management
- Environmental constraints (technical, geographical, human, natural)

Although the Dynamics CRM platform supports rapid and agile application development, the following should be considered and reviewed prior to progressing with the development phase:

- Application parts
  - An application can comprise multiple features
  - A feature may comprise multiple components
  - A feature may depend upon other features (at a component level)
  - A feature may have dependent features (at a component level)
- Application packaging
  - Wrapping CRM customization, assembly, and data deployment as part of a larger, manual or automated deployment process
    - Leveraging CRM solution management framework to produce deployable CRM Solutions
- Application Deployment
  - Managing the process of moving components of an application between environments in a simple, repeatable and efficient manner
    - Leveraging automated deployment tools for Dynamics CRM between environments
    - Dedicated role for package management, building the application setup package from the start of the development phase
  - Planning the release calendar of the packages, planning and mitigating changes
  - Provisioning development, test, acceptance and production environments
- Application Support
  - Version management process of specific environments
  - Planning and managing development, service pack, and hotfix branches
  - Quality assurance gates for the entire project lifecycle (quality testing, code reviews, check-in reviews, code change analysis)

The remaining sections in this document will provide the guidelines and best practice recommendations to manage these areas. The most appropriate approach may differ between customer project scenarios.

Lifecycle Management Overview

ALM encompasses the whole delivery lifecycle from envisioning and requirements gathering through design, development, and test, culminating in deployment and subsequent operation. ALM builds on three pillars to manage delivery activities:

- Support, track, validate and enforce the specific development processes for delivering application artifacts
- Document, manage and track the relationships between development and delivery artifacts and the specific activities producing them
- Report on the progress, quality and possible risks of the delivery

The enterprise-level ALM process covers the three main areas detailed in the following table:

Area	Detail
Governance	▪ Business case management ▪ Project portfolio management ▪ Application portfolio management ▪ Technology mapping, POC labs ▪ Decision support
Development	▪ Development methodology (SDLC) ▪ Development team management ▪ Scope, time, resource management ▪ Hotfix, version, branch management
Operations	▪ Deployment ▪ Versioning ▪ Monitoring ▪ Updating, maintenance ▪ Upgrades/migration

For guidance regarding the governance aspect of Dynamics CRM implementations, more information can be found in the Dynamics Sure Step methodology pack about methods, samples, and project setups. General reference information about the existing methodologies can be found in Appendix B: Methodologies.

The operations aspect of Dynamics CRM infrastructure can be found in the Dynamics CRM Implementation Guide, which covers the planning, deployment, maintenance, and administration of Dynamics CRM. See the Introduction to the Microsoft Dynamics CRM Implementation Guide for further details. In addition, the Deployment section covers the specific operational details from a solution management perspective.

Considering the solution development lifecycle aspects of enterprise Dynamics CRM, projects will require the development methodology to cover the following areas:

- Requirements management
- Build management
- Version control
- Bug tracking
- Test management
- Reporting
- Team development, team management
- Source control, branching
- Command line builds, daily builds
- Localization
- Code reviews, QA processes

Dynamics CRM deliveries vary from typical enterprise application deliveries in the area of platform use. While this fact incurs a level of constraint in terms of the approaches that can be used to edit, build, and deploy components, the version management strategies for enterprise-level CRM solution development and the corresponding environments are similar to any enterprise application and can be characterized as follows:

- Need for a single solution or multiple solutions
- Need for single or multiple consecutive releases or need for parallel releases
- Single or multiple feature development teams
- Need for layered solutions with common solutions providing re-use across multiple sites
- Creating complex applications containing CRM, web applications, custom or external component/service solutions with common or separate versioning

The following sections will highlight the recommendations and best practices for delivering enterprise Dynamics CRM applications using an enterprise delivery framework throughout the entire development lifecycle.

Scenario

A single scenario will be used throughout this document to facilitate understanding of concepts presented. The scenario reflects a complex, enterprise environment that allows comparisons to be drawn to approaches that might be used by an ISV or global implementer.

Contoso Health Ltd. is a market leader in the field of medical test equipment solutions. Contoso engages in all market segments, from laboratory-based professional devices to self-test consumer devices. As part of its forward thinking IT strategy, Contoso has decided to implement all client-facing applications on the Dynamics CRM platform.

The customer base and products vary significantly; depending on the target market segment, products range from small consumer blood glucose monitors in diabetes care to large blood analyzer systems used for professional diagnostics in laboratories and hospitals.

Consumer Device: handheld blood glucose analyzer

Professional Device: laboratory blood analyzer

Several business groups cover the various market segments, but common to all are the following functional areas:

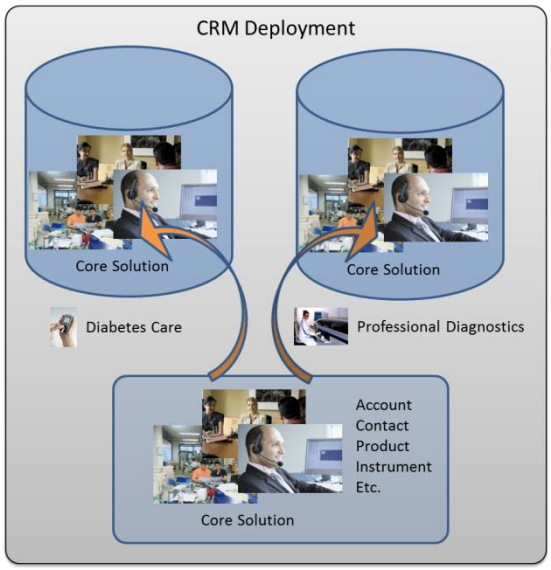
Sales Force Automation

Call Center

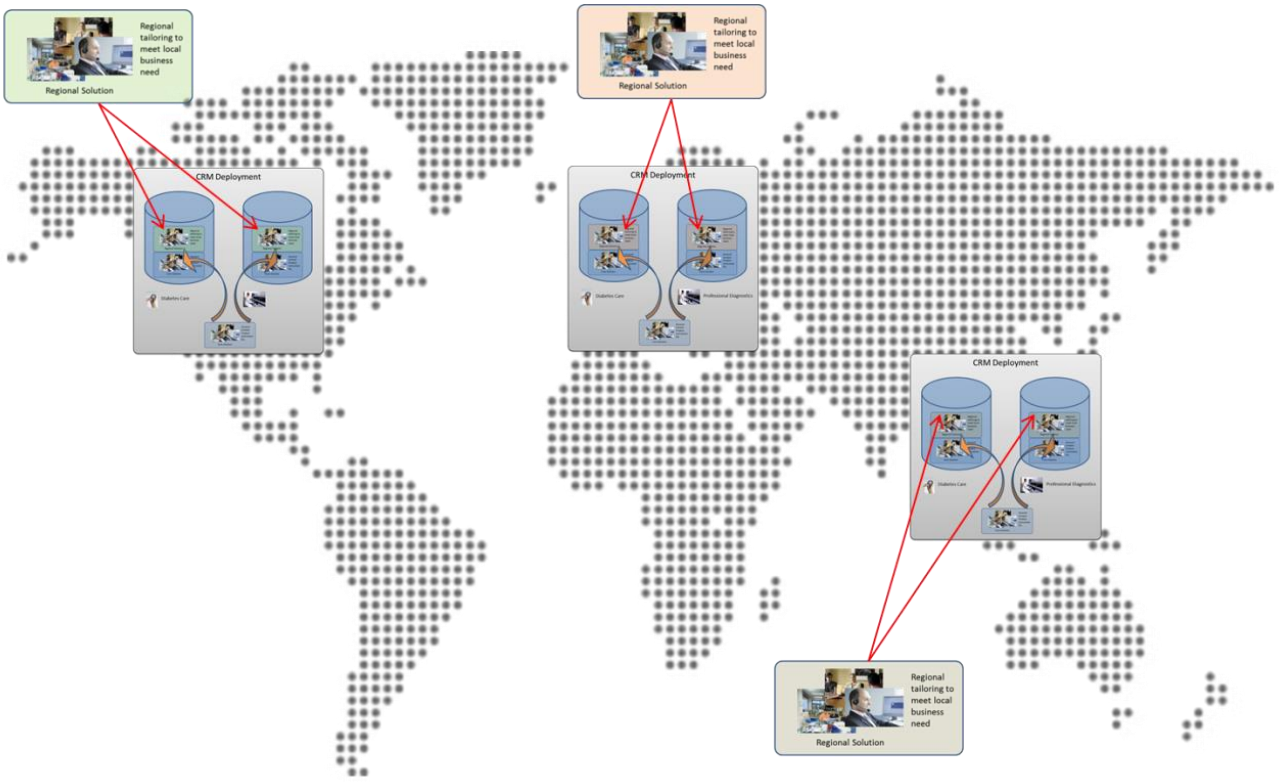
Field Service

It has been determined that the core requirements of each market segment can be met by a single application that may be deployed multiple times, once for each business group that requires it. Most of the differences relate to the data that will be contained within the system (product lists, suppliers, customers), but a degree of tailoring to the individual market segment will also be supported to ensure functional and regulatory requirements can be met.

The following diagram provides an illustration of this approach.



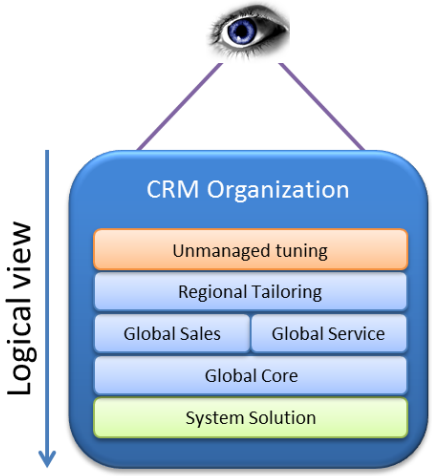
As a market leader, Contoso Health Ltd. has subsidiaries on three continents. To ensure fast response times, the company has decided to provide regional deployments of Dynamics CRM, with each subsidiary deploying the CRM solution locally per market segment they cover. Physically, there may be two deployments per region to support longer-term variance between the market segment applications. In addition to the flexibility to tailor functionality by market segment, Contoso also requires the ability to tailor the functionality of the core solution regionally due to the fact that each subsidiary enjoys a level of autonomy in terms of the processes they implement.



With the ever increasing velocity of technology change, Contoso realizes that it needs the flexibility, agility, and choice in how, what, and when the company deploys functionality to its regions. This is evidenced in APAC where functionality to enable its sales force is all that is initially required but with a time criticality due to a competitive market. This, however, results in a need for the sales capability to be delivered in isolation prior to the availability and completion of the call center and service functionality. At a point in time to be determined, APAC may additionally require these features.

To support this structure and rollout, it has been decided to take a layered, multiple-CRM solution approach, breaking down the application into a number of constituent parts for isolation and ease of reuse. The “Global Core” solution will need to isolate functionality that supports both service and sales. Further discrete global functionality, for service and sales respectively, will be provided within two further global solutions. One or more regional solutions will be layered on top to tailor for market segment and/or sales versus service functionality.

Note: In terms of Dynamics CRM solutions, layering occurs when a specific component is affected by change within one or more solutions. The following illustration provides a representation from a logical perspective of the solution layers constructing the complete application for a market segment within a regional deployment.



- The logical view illustrates that although physically only one solution package may exist in a layer, at a component level (due to no overlap in solutions) both the global sales and global service coexist logically at the same level.
- Regional tailoring may be broken into 2 solutions to isolate service tailoring from sales tailoring regionally
- Fine tuning may be performed directly in the unmanaged layer

Requirements Summary

The Scenario articulates a variety of business and IT requirements, which can be distilled into providing the ability to:

- Develop a centralized application that can be deployed regionally
- Enhance the application functionality based on regional requirements
- Maintain the centralized application independently of regional enhancements
    - Use a flexible and agile deployment methodology that provides for isolating and segmenting features for the purpose of reuse, stability, and functional requirements

From an IT perspective, there are additional, less explicit requirements that include providing the ability:

- For multiple feature development teams to work together on application development in a controllable and scalable manner
- For each team member to work in a sandbox to avoid blocking other team members
- To granularly version control the components of the application
- To support parallel version development and hotfixing

CRM Solution Lifecycle Management

When implementing a business application on the Dynamics CRM platform, a number of key topics that must be considered. These topics are similar to those that would apply to any enterprise software application:

- Application Components
  - An application can comprise multiple features.
  - A feature may comprise multiple components.
  - A feature may depend on other features (at a component level).
  - A feature may have dependent features (at a component level).
- Application Packaging
  - Features and components of an application should be combined into packages for version traceability and deployment.
- Application Deployment
  - Deployment should be repeatable and predictable.
  - Deployment should be reversible.
  - The Deployment process should facilitate the hotfix process.
- Application Support
  - The application should be designed for operation, ensuring that ongoing live operation of the application is manageable.

Dynamics CRM applications tend to differ from generic enterprise applications in their platform-based nature. This nature is ultimately a strength in terms of speed of delivery and quality, but it results in behavior that needs to be fully understood regarding the entity model and, specifically, regarding the bonds to the data schema and also the form-based user interface. Dynamics CRM is the editor for and consumer of the entity model and is also the gatekeeper for the solution framework that controls how different solution packages interact with each other.

CRM Solution Concepts

Customizers and developers use solutions to author, package, and maintain units of software that extend Dynamics CRM 2011 and Dynamics CRM Online. Solutions are distributed to organizations so that they can use Dynamics CRM to install and uninstall the business functionality defined by the solution.

Solutions and layers are fundamental concepts within CRM that need to be fully appreciated and understood to construct an approach to lifecycle management for Dynamics CRM applications. There are three key concepts that need to be introduced:

- Solution Packages: Act as containers for functionality required to be deployed as a unit
- Layers: The consequence of a specific component being affected by change within one or more solutions
- Managed Properties: The mechanism to control how layers interact with each other
  - Configuration of the level of lock down; controlling the degree of further customization that is possible to components within a given solution package

These terms will be used throughout the remainder of this document and should be fully understood. For additional detail on these terms and how these concepts interact to provide the solution framework within Dynamics CRM, see Appendix D: CRM Solution Concepts and the Dynamics CRM 2011 SDK section titled Package and Distribute Extensions.

Solution Management Processes

To create the components of a CRM solution, it is necessary to work within the unmanaged layer. In solution terms this is analogous to working with source code. Continuing the analogy, source code would be compiled into binary form prior to deployment to production, and this is similar to the export of a managed solution.

From an enterprise development perspective, it is beneficial to test the deployment process for production as early and as often as is possible. Consider the following environments:

UAT/Staging environment(s) should be architecturally and functionally representative of the production environment. The approach to deployment within these environments should be representative of the deployment to production. Therefore, a managed solution would be deployed to these environments in the same manner taken for production. Continuing this logic backwards, the approach to deployment within the CIT environment should be representative of the deployment to UAT. The earlier within the cycle the final deployment approach can be tested, the more opportunity there is to resolve issues and ensure repeatability and predictability. To support this approach every environment downstream of development should deploy managed solutions.

Note: For additional detail about the solution management process, see the Deploying Microsoft Dynamics CRM 2011 and CRM Online Solutions from Development through Test and Production Environments white paper.

The Lifecycle from a CRM Solution Perspective

An application for Dynamics CRM consists of one or more Solution Packages that will be deployed into a CRM organization. The deployment processes (initial version, hotfixing, next version) need to be repeatable and predictable. From a CRM Solution perspective, these processes must be managed irrespective of whether Source Code Control may support the approach. The following scenarios consider these processes purely from the Dynamics CRM Solution perspective.

Note: Maintaining a complex menagerie of solutions and the dependencies between them is an important role within Dynamics CRM programs and for ISV solution library providers. This can consume a significant amount of time and should be considered up front when planning projects and programs.

The Hotfix Process

A *hotfix* is a single, cumulative package that includes one or more files that are used to address a problem in a product/application and are cumulative at the binary and file level.

The approach to hotfixing a live deployment varies and depends upon the type of amendment being made. A sound understanding of the Solution Framework and the detail provided in the CRM Solution Concepts section of this document is required.

While most hotfixes are amendments to existing components, a variety of hotfix types might be called for, including:

- Amendments to existing components, for example:
  - A bug fix in a JavaScript web resource
  - A change of column display order within a view
- Creation of new components; for example:
  - A new configured workflow process to set a state flag that was previously forgotten
  - A new plug-in to control a business process
- Deletion of existing components; for example:
  - Removal of a view that should not have been made available

As a result of the additive nature of metadata changes within Dynamics CRM, hotfixes that require amendment or creation of components are relatively straightforward; however, deletion of components is more complex. These are considered in depth below.

Hotfix by amending or creating a component

Note: When a hotfix requires the amendment of an existing component or the creation of a new component, the most effective and reliable approach is to generate a minimal solution. For this approach to work, the solution must maintain the same solution publisher and the same solution name (see note below).

Using the example presented in the Scenario, the method to create the minimal version of the "core" solution will require a new organization to be created. This is necessary as solution names are unique within an organization database, and this solution will contain a subset of the components maintained within the "full" core solution. The changes to the existing components or the creation of the new components can be undertaken against the original development organization. An unmanaged version of the core solution can then be exported from this organization into the new, transient organization. Following import, all components that have not been affected and, therefore, are not part of the hotfix can be removed from the core solution within the transient organization. This process results in the generation of the minimal solution.

There are two variants to how the hotfix should be applied to a production environment:

- Maintain the version number by overwriting the existing solution (only the affected components)
- Import with an incremented version number of the existing solution and create a new layer

Using the recommended approach to preserve customizations when importing managed solutions, maintaining the version number will automatically replace any components within the solution being imported. Conversely, incrementing the version number will create a new managed layer for the solution above the old layer.

Option 2 can become misleading, however, as it suggests that the hotfix can be uninstalled or deleted which is not necessarily true due to the additive nature of the platform.

The weakness of Option 1 is the lack of ability to identify that a hotfix has been applied. This can be overcome by adding a note of the current hotfix level within the description field of the solution.

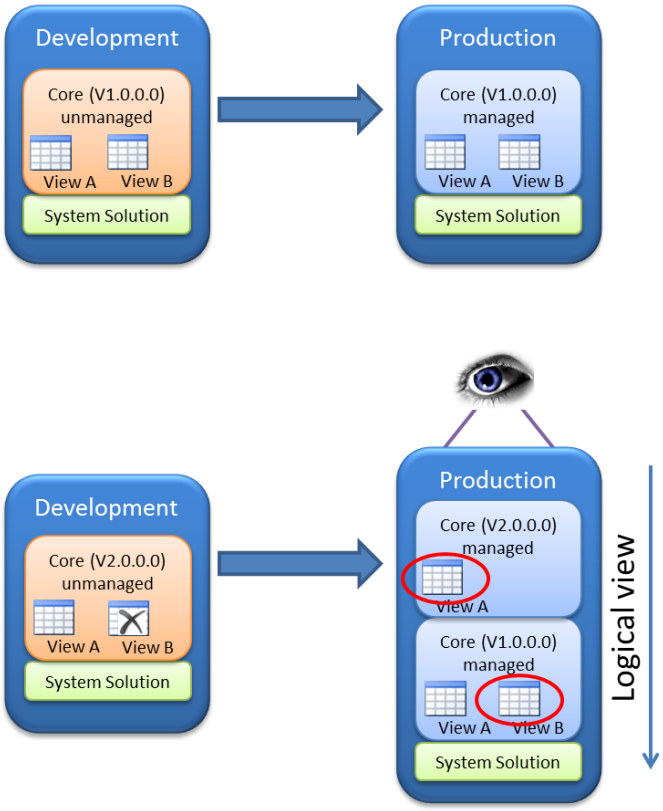
Note: Simply using a different name for the solution is not reliable. Considering the example presented in the Scenario, for which the core solution needs to be hotfixed, if a new solution is created with a different name using the same publisher, the process will appear to work in the intended manner until the next version of the core solution is deployed. At this point, selecting the option to import without overwriting will import the new version of the core solution within a layer directly above the previous version of the core solution. This will be beneath the solution containing the hotfix, which therefore may result in configuration and customization changes not being surfaced.

Hotfix by deleting a component

Solutions (managed or unmanaged) only contain references to the components they require. Using the detail in the Shared Publisher Technique section below, it is possible to remove a reference to a component within a managed solution via a transient solution.

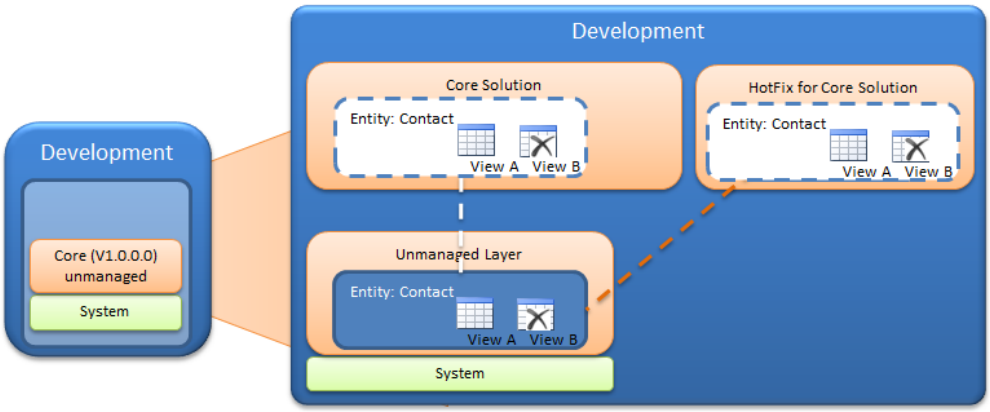
Consider the Scenario above, assuming that Version 1.0.0.0 of the Global Core solution has been deployed for two market segments within Europe: Diabetes Care and Professional Diagnostics, respectively. Following go-live, it is identified that a view for the contact entity is not relevant to both market segments and, therefore, should not exist within the core solution as it exposes data about service contracts within the consumer diabetes care application.

Given that metadata changes are additive, there is no straightforward method to remove the view from the database. The contact entity is a system entity and cannot be deleted; deploying Version 2.0.0.0 without the view will not remove it since version 1.0.0.0 will still contain a reference to the view as can be seen in the following diagram:



To remove the view from the core solution, all references to the view across every version of the core solution must be removed. This can be achieved as previously mentioned via a transient solution and the shared publisher technique. The following diagram represents the internals of the core solution and demonstrates that the two custom views for the contact entity reside in the unmanaged layer.

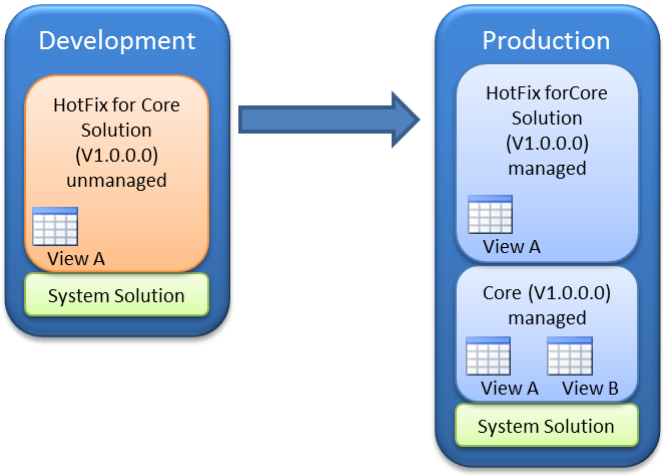
Shared Publisher Technique



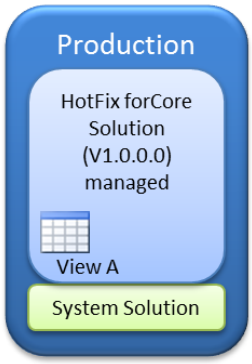
The core solution references these views through its reference to the contact entity. Additionally, any further components that are required by the core solution are also referenced. A new solution for the hotfix is created using a unique solution named “Hotfix for Core” solution while maintaining (sharing) the same solution publisher used by the core solution. All entities referenced by the core solution are added as references to the “Hotfix for Core” solution.

The view that is no longer required is deleted from the unmanaged layer which removes the references in both the core and “Hotfix for Core” solutions.

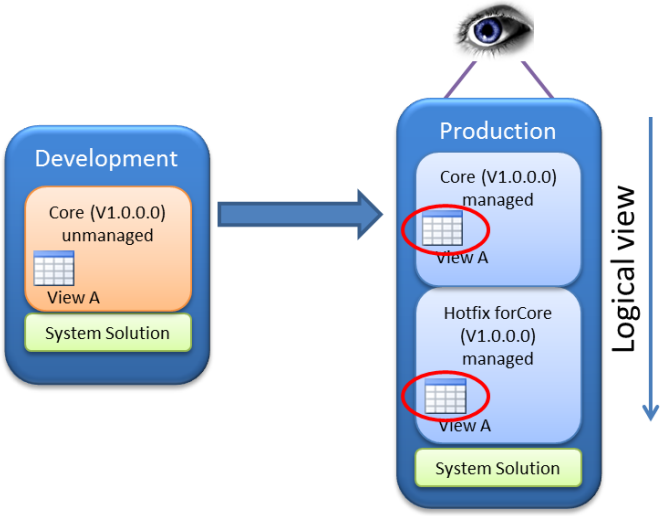
The “Hotfix for Core” solution is then exported as a managed solution and imported into the production environment.



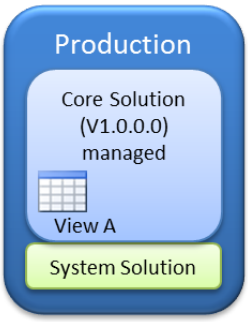
The “Hotfix for Core” solution contains references owned by the same publisher to all the same components as the core solution, except for the view to be deleted. Because of this, it is possible to delete the core solution, which will only delete components that have no remaining references from the publisher (that is, View B).



A new managed version of the core solution can now be exported from development without including the view and imported into production.



Since the core solution contains references owned by the same publisher to all the same components as the “Hotfix for Core” solution, it is possible to delete the transient “Hotfix for Core” solution without the deletion affecting any further components.



Supporting One CRM Solution for One Production Deployment

The simplest requirement for any Dynamics CRM Application is to be able to effectively manage the process of deploying a single CRM solution to a single CRM organization. Once live and in production, there may be future versions of this CRM solution that need to be deployed into the live environment. Additionally there may be a requirement to hotfix the current live version of the Dynamics CRM solution while the future version(s) are still under development.

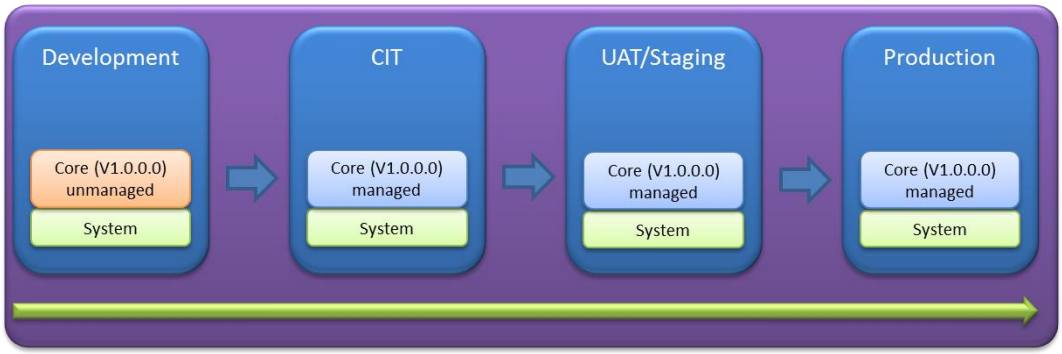
Considering the illustration for Contoso in the Scenario section above, at the heart of the CRM applications the IT department need to deliver to their business is a core solution. The core solution provides typical account, contact and product-based information and, in general terms, may be considered to contain:

- Entity configuration (including any custom entities)
- Site map and ribbon changes
- Web resources including Silverlight components
- Processes
- Plug-ins

Version 1 Deployment

Purely from the CRM solution lifecycle perspective, there are no major challenges to deploy version 1 of the “core” CRM solution to production.

- The solution takes no dependencies on other CRM solutions apart from those satisfied by the system (out-of-the-box) solution which is always present.
- Deployment of the managed version of the core solution propagates through the environments, ultimately going live in production after satisfying all gate criteria.



Version 1 Patching

As development begins on version 2 of the core solution, it is also important to support simultaneous development of hotfixes. Doing so enables the process of patching defects identified in the current production system. A minimum of three CRM organizations are required within the development environment to support two versions of the solution:

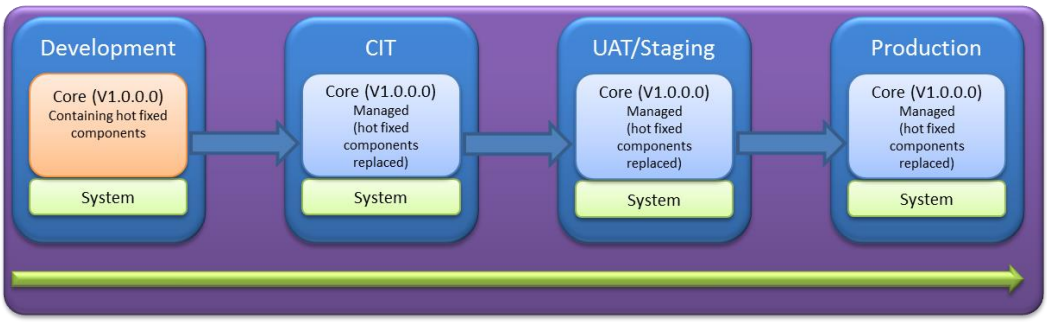
- Version 1 + existing hotfixes
- Current hotfix development (potentially a transient organization)
- Version 2 development

It may be necessary to extend the minimum requirements of the development environment to include two CRM deployments. These may be required for situations where different CRM Update Rollup versions exist between production and the development environment or where version 2 development needs to take a dependency on features from a later platform release.

Note: While it is necessary to be able to reconstruct what is currently deployed in production for the purpose of further hotfix creation, having the Dynamics CRM organization permanently available may not be required. The organization could be transient to support the hotfix process. However, consider this carefully to ensure that urgent production issues can be addressed adequately and in a timely manner. Virtualization and snapshots/resets can significantly support the process of generating transient organizations and subsequent teardown.

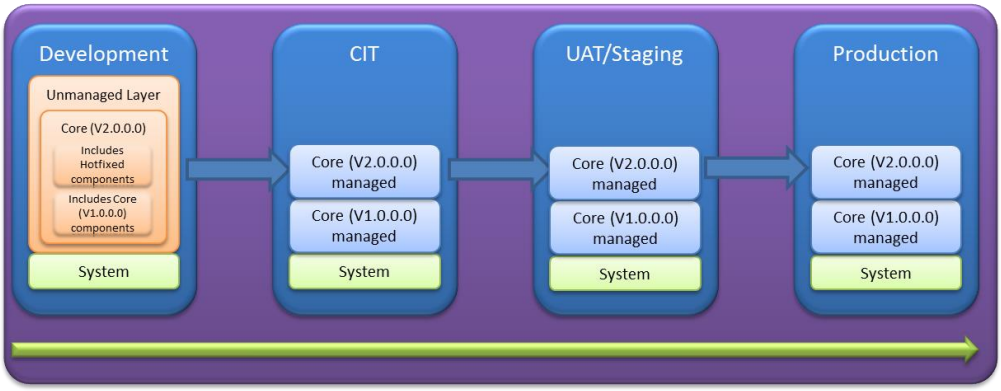
As previously noted in the Hotfix Process section, the exact method of development and approach to deployment of a hotfix depends on the nature of the hotfix – whether it is an amendment or a deletion. Taking the logical approach for the purpose of illustrating the solution lifecycle denoted above, a minimal version of the core CRM solution containing only the changed components of the version 1 core solution is constructed. This uses the same version number to replace components rather than generating a new layer on top of the original solution.

The hotfix for the core solution propagates through downstream environments in a manner similar to that of the core version 1 solution. Additionally, the hotfix would need to be applied to the CRM organization containing version 2 development work in progress as an unmanaged solution.



Version 2 Deployment

Version 2 of the core solution propagates through downstream environments in a manner similar to that of version 1 of the core solution and the hotfix for the core solution.



Logically, version 2 of the core-managed solution package is essentially a superset comprising:

- Functionality of core version 1.
- Hotfixes for core version 1.
- Additional functionality of core version 2.

In situations where features have been deprecated between versions of a CRM solution (for example the removal of a view) there is a little more complexity. Logically, version 2 of the solution is a superset as described above, although in this case it is a negative addition and the deployment process requires use of a transient solution with the Shared Publisher Technique to produce the same end result. See the Hotfix Process section for full detail.

Supporting One CRM Solution for Multiple Regional Deployments

Reflecting on the Scenario section, it is apparent that the core CRM solution would be reused by a number of business groups to cover their market segments (in the example, Diabetes Care and Professional Diagnostics as a starting point). This model would then need to be replicated across Contoso's regional grouping of subsidiaries through a regional deployment model.

From a CRM solution perspective this approach appears to fit well and this is true when considering the simple case, deploying the same version of a CRM solution multiple times for multiple deployments of Dynamics CRM. In Contoso' scenario, they would simply deploy the global core CRM solution twice in each region. For each region, the CRM deployment would host two organizations, one for the Diabetes Care application and one for the Professional Diagnostics application respectively. In total, this would result in six deployments of the same version of the global core CRM solution.

The most complex variant of the above that could occur over time is for each organization within each regional deployment to be on a different version of the core solution. This would result in a need to support in parallel six versions of the core solution, requiring a minimum of six organizations within the development environment. Additionally, within the development environment there may be a requirement for multiple CRM deployments to support the six organizations. This will depend on the Update Rollup level and platform version for each physical regional deployment.

In reality, it is unlikely for the most complex variant to occur, particularly in an enterprise environment where deployment can be controlled through mandated upgrades to the same version of the solution. However, it is likely that due to business or regulatory requirements, the application functionality required within a market segment may diverge from another market segment. This may require functionality delivered through an Update Rollup or biannual release that necessitates the regional deployment being split into two physical deployments to support the variance in platform version required. It is also likely that one region may adopt newer versions of the application at a faster rate than that of another region.

In summary, there is a need to support multiple versions of the same solution. This is no different to the approach detailed in the Supporting One CRM Solution for One Production Deployment section except the approach may be further extended to support a greater number of concurrent versions.

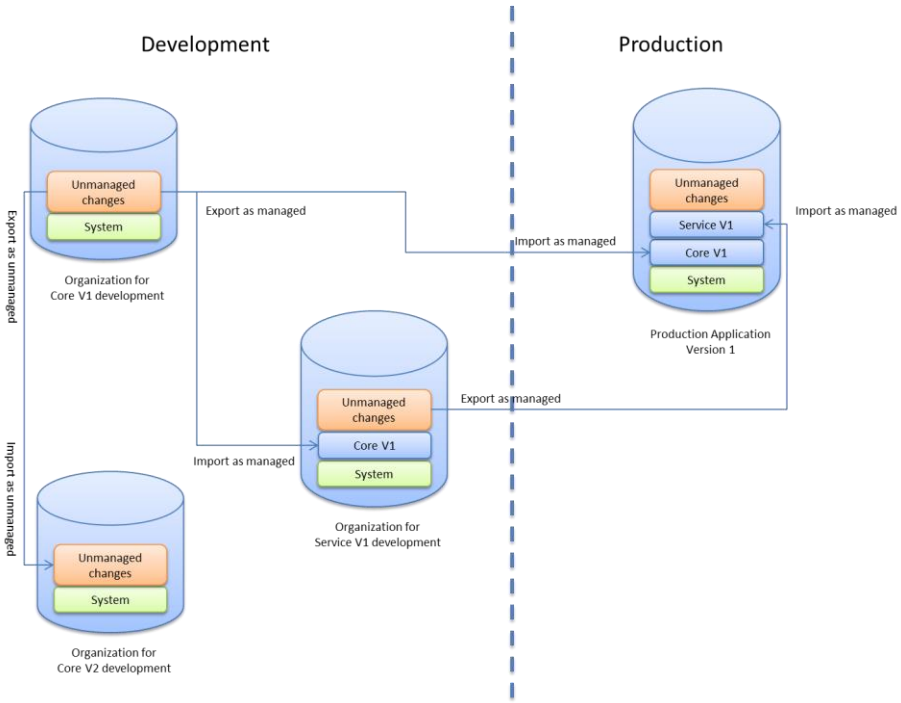
Supporting and Managing a Multi-Solution Environment

An application that comprises multiple Dynamics CRM solutions requires an extension in sophistication of the approaches previously detailed in this chapter to support and manage an application consisting of a single CRM solution. The initial perspective to consider is one where a multiple CRM solution application is simply a grouping of a number of single CRM solutions. Using this perspective enables the rules defined in the previous section to remain applicable when considering the management of each solution independently.

Where it becomes more complex is when one CRM solution needs to take a dependency on another. Unmanaged components may be considered analogous to source code in a typical software application. Managed solutions are the analogy of assembly or binary references where a component needs to take a dependency on another component.

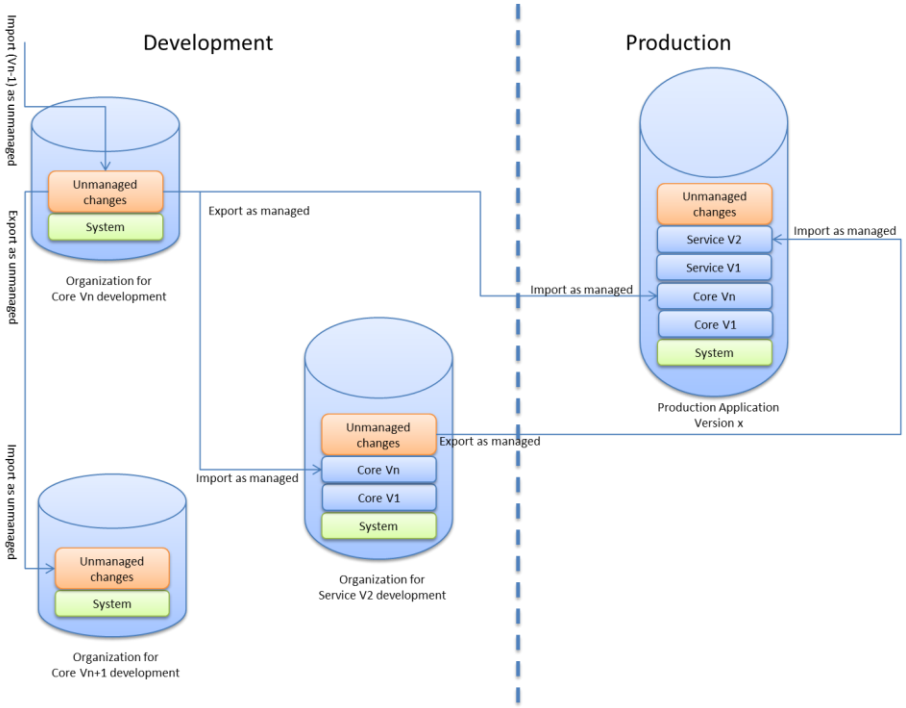
The single solution application introduced the requirement for a CRM organization per version of the solution that was being maintained and developed. This is also the requirement for the managed solution from which a dependency will be taken in addition to the organizations required for the dependent solution.

Taking the example from the Scenario, it has been demonstrated how the core solution will be supported and deployed to multiple regions. An additional solution, dependent on the core solution, will be created and maintained centrally for the service-specific features. From a logical perspective, it is simplest to consider the development environments for the core and service solutions as being distinct and isolated. Physically, these may consist of one or more organizations per solution hosted on the same CRM deployment (subject to Update Rollup and platform version dependencies).



When a new version of the core solution is available (and required) by the service solution, it is imported as a managed solution into the development organization for the service solution. This allows the new version of the core managed solution to layer directly above the previously imported version and, importantly, beneath the unmanaged components of what will become the service solution.

Note: The dependencies on the core solution are at a component level not at a solution level.



The layering and solution dependency model can extend to involve additional CRM solutions and versions by incorporating further managed solution dependencies and organizational databases to support their development.

Important: When a dependency is taken on a CRM solution, the development organization for the dependent solution should be treated as if it is production – from the perspective of the solution that a dependency is taken on.

Note: It is always recommended to restrict changes to components within a solution as much as possible through Managed Properties. While these constraints may be relaxed at a later point in time, they cannot be tightened easily (see the *Managed Properties* section of Appendix D: CRM Solution Concepts for further details).

The Lifecycle from a Development and Version Control Perspective

An application for Dynamics CRM consists of one or more Solution Packages that will be deployed into a CRM organization. The deployment processes (initial version, hotfixing, next version) need to be repeatable and predictable. Using development and version control techniques will support and simplify the approaches required to manage these processes. The following scenarios consider these processes and how they augment those presented purely from vantage point detailed in the Lifecycle from a CRM Solution Perspective section.

Throughout previous versions of Dynamics CRM, a typical approach to the management of CRM development was to store the exported customization file “as is” within a version control system. Parallel development of customizations was difficult due to the need for one or more instances of Dynamics CRM and multiple, distinct organizations to support parallel work.

With the introduction of the Dynamics CRM 2011 solution framework concept, the packaging and layering of specific components is now supported; however, the process of managing parallel development and tracking customization changes can still be a challenge on enterprise Dynamics CRM projects. The following sections cover the different aspects of version control challenges with respect to Dynamics CRM solutions and provide recommendations and best practices for the approaches to address them.

Note: Maintaining a complex menagerie of solutions and the dependencies between them is an important role within Dynamics CRM programs and for ISV solution library providers. This can consume a significant amount of time and should be considered up front when planning projects and programs.

Requirement and Work-Item Tracking

Typical development processes require various types of information: requirements, tasks, bugs, tests, etc. Central management of this information is crucial for enterprise CRM development. The Visual Studio environment and Team Foundation Server (TFS) work-item tracking can easily support enterprise-level CRM development of any size, storing and connecting development work-items. To fit the needs of an organization or engagement, TFS also supports full customization of the built-in development artifacts and processes.

Dynamics CRM deliveries using the Dynamics Sure Step methodology will typically produce a base WBS as the output from the design phase. TFS work-item schema and the Excel import-export feature can be used to import the requirement list and create the initial WBS even on larger CRM projects.

Using TFS work-item tracking, you can leverage the standard features for:

- Requirement lists and change management.
- Work-item administration.
- Team management and work-item allocation.
- Managing areas and iterations.
- Managing test cases and scenarios.
- Managing bugs and resolution lifecycle.
- Tracking relations and dependencies between requirements, tasks, and change impacts.
- Standard reporting of work progress and product quality.

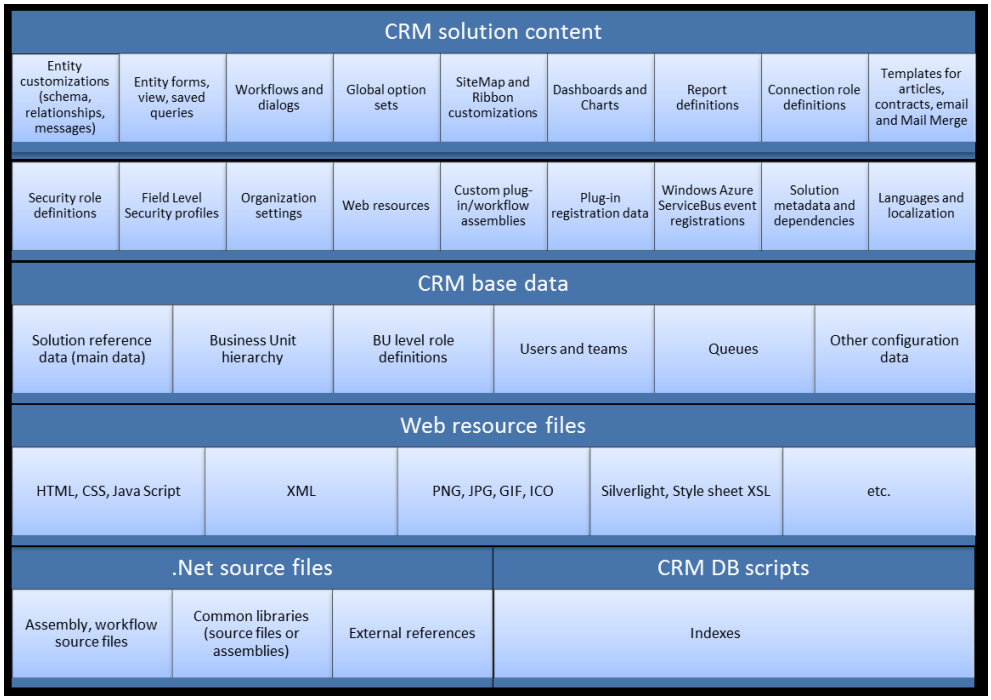
Version Control

One of the most fundamental challenges in any enterprise development project is the control, tracking, and management of changes within source code files.

From a CRM/XRM solution perspective, the following items can be considered as source elements:

- CRM solution content
- CRM base data
- Web resource files
- .Net source files
- CRM database scripts

The detailed content of the above elements is illustrated in the following figure:



The Dynamics CRM solution content is stored and versioned by Dynamics CRM 2011 in a single binary solution zip file.

Deciding How to Version Control CRM Solutions

The Dynamics CRM solution package may be stored as binary or uncompressed format under version control. The binary approach provides no opportunity to compare differences between versions, while storing uncompressed files enables only the following individual CRM source elements to be independently version controlled and tracked:

- Customizations.xml
- Solution.xml
- Workflows
- Web resources (built by the web resource file projects)
- Plug-in assemblies (built by .Net projects)

All other elements in the customization and solution XML require manual creation or merge into one single source file by a developer as necessary. The complete CRM application may be divided into a number of distinct CRM solutions to support more granular version control for specific modules or functionality.

Using Unpacked CRM Solutions

To be able to store and version control the individual components of CRM solutions, the XML files need to be split into basic elements and stored discretely under version control. This approach supports the individual tracking and change control of components within a CRM solution. The unpacking of solution XML files need to be executed for every version change to CRM configuration and tracked via version control. In a developer environment, the unpacked solution would need to be packed together with any binaries built in order to create a CRM solution to deploy within their development environment. The packing operation is also required during a team build to create the CRM solution package for deployment.

SolutionPackager is a tool that can reversibly decompose a Microsoft Dynamics CRM compressed solution file into multiple XML files and other files so that these files can be easily managed by a source control system. For more information, see Appendix E: Managing Complex CRM Scenarios by Using the SolutionPackager Tool.

Bringing Together with External Components and Work-Items

In addition to the components that comprise a Dynamics CRM solution, enterprise CRM projects typically contain a broader set of source components which also need to be version controlled:

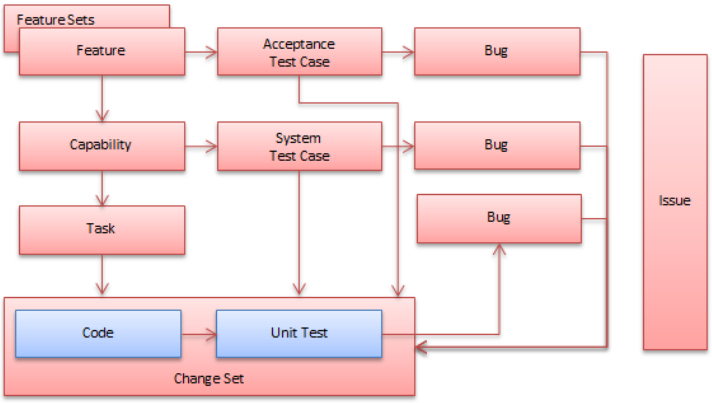
- Unit tests
- Setup scripts and solutions
- Deployment scripts
- System test and automated test scripts
- Integration and migration scripts (SSIS, PS, …)
- External tools, SDK, etc.
- All other custom components with source, test and setup projects
  - Custom DB projects
  - Web application projects
  - Service projects
  - WPF/WCF/console projects, etc.
- The build definitions to all of the above

The individual storage of all custom sources and CRM customization elements makes it possible to track the requirements to individual code changes and customization change-sets connected to specific bugs. This granularity supports approaches to provide the quality check gates for tracking an individual developer's work and the level of changes within the entire source tree.

Using binary or full CRM solution file-versioning techniques, only entire CRM solution package versions can be bound to specific requirements, work-items, bugs, and change-sets. Reviewing and rolling back individual changes is extremely difficult, requiring manual intervention. The rate of customization changes between two CRM solution versions can only be tracked using manual comparison.

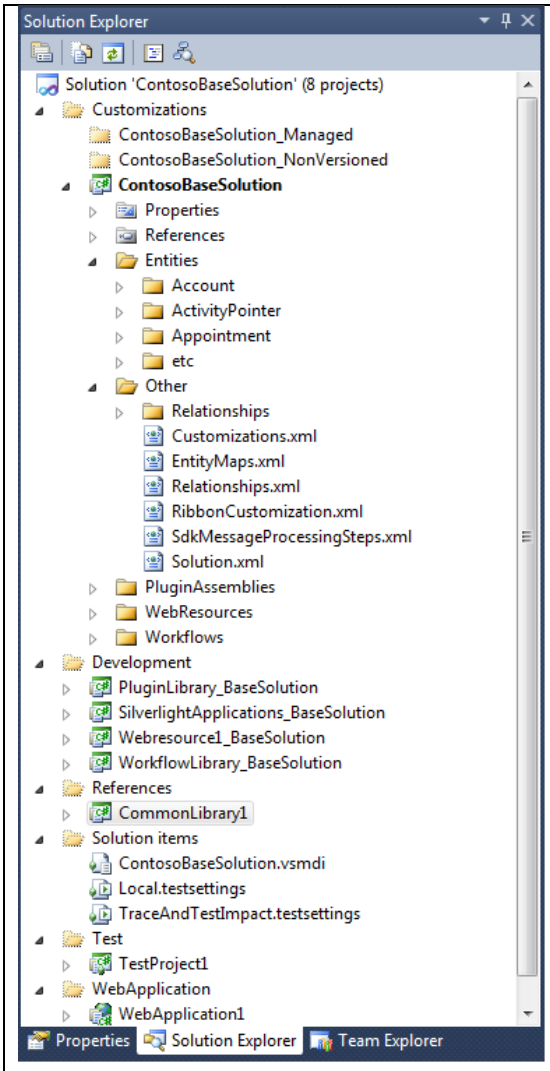
Using the unpacked solution tree approach, specific individual changes can be bound to work-items or bugs using smaller change-sets. Reviewing and rolling back individual changes is far less complex and the rate of changes (code churn) in the CRM customization tree will be measurable automatically.

A sample dependency tree between requirements, tasks, check-ins, and bugs is illustrated in the following graphic, which is and is taken from the Iterative Solution Development (ISD) methodology - standard work item relationship mapping explained in the Iterative Solution Development section of Appendix B: Methodologies.



Sample Version Control Tree

The sample version control tree for an enterprise Dynamics CRM application, supporting multiple solutions, is illustrated in the following figure:



One Visual Studio solution for all CRM application elements and components

Customizations

- One folder for CRM solutions versioned as binary or full xml format (managed or unmanaged format)
- One project per CRM solution using unpacked format (see Appendix D: CRM Solution Concepts)

Development

- Custom developed components of CRM customizations – separated per CRM solution
- Plugin library projects (multiple libraries may be used per solution)
- Silverlight application projects
- Web resource projects
- Workflow Activity projects

References

- Common components
- External references
- As binary or built during the daily build

Solution items

- Test settings and definitions
- Other VS solution elements

Test

- CRM unit test projects
- Custom Development unit test projects
- UI and automated test projects

Web Application

- Custom Web Application projects
- Typically deployed separately or in ISV folder (deprecated)

Managing Versioning Requirements

For Dynamics CRM projects in enterprise environments, there are a number of perspectives that need to be satisfied with respect to the approach to versioning:

- Supporting multiple parallel versions of the application in production and development
- Supporting parallel development branches for short and long term
- Supporting development projects comprising multiple phases spanning multiple years
- Supporting a solution management process that enables daily build and deployment
- Consistent versioning across all solutions
- CRM Solution layering – supporting solution fixes and updates, and multiple additional feature pack solutions
- Ability to deploy the solution to multiple environments
- Ability to easily track changes in CRM configuration and customization, covering changed values, modification dates, and modifier

When selecting the appropriate version control method, be sure to consider the following approaches.

Storing the CRM Solution as a Single File

- This method favors the one CRM solution to one developer topology – isolating functional areas into separate solutions.
- If multiple developers are working with the solution, it is typical to have a central configuration organization – shared by the developers. The daily version control process when using this method is typically executed manually. The person nominated as configuration lead or build master manages the central configuration CRM instance. This instance serves as the central repository of all CRM configuration for the solution. Unmanaged versions of the CRM solution are exported and placed under TFS (typically via association to a Visual Studio solution) at some predefined intervals (for example, once/twice per day, just before build, on completion of configuring an entity, etc.).
- Parallel versions of a solution may be supported, but it requires manual management of possible scenarios (illustrated in the Lifecycle from a CRM Solution Perspective section). Managing parallel versions of complex solutions is not recommended in this way.
- Support of long-term, parallel development branches is not recommended because the configuration merge can only be executed on a CRM environment manually. In these cases, it important to merge branches frequently and to integrate code onto a combined branch (or back to the trunk). Another approach is to create two entirely unrelated CRM Solutions and configure the changes in both – doubling the work effort.
- Supporting long development projects which may span multiple years is feasible by creating layers of CRM solution versions regularly, limiting the change tracking requirement under version control.
- When using this method, the daily build process is responsible for integrating the unmanaged solution package with the compiled web resources, assemblies, and plug-in registrations. The build is essentially an online process as it requires a running CRM instance. The CRM instance will integrate all the elements together and will also export the entire package as both managed and unmanaged CRM solutions.
- Consistent versioning of all elements within the CRM solution can be implemented by extending the build automation of the Developer Toolkit. However, this process should consider the number of build agents and how to uniquely identify the CRM instance and organization to use for the build (particularly when supporting multiple build definitions). The CRM solution XML file always contains the strong-named, fully qualified name of each assembly. These need continuous updates with the latest build versions, requiring the build process to manage registration within the CRM solution that will ultimately be exported from CRM as the drop from the build.

- Tracking individual customization changes, work-items, check-in dates, modifiers, and customization history is almost impossible. TFS file-based differencing will fail on the large size of the customization.xml file. It is possible to track that a change was made in the solution customization and the date and time that it occurred but only with minimal fidelity. There will be no information about the specific change unless a full manual compare is conducted even if the change was only a new label, new form, new entity, or even a cascade setting on a relationship.
- Tracking specific changes and their time of occurrence in the customization tree is not possible (for example, tracking who changed a relationship cascade setting). The built-in Annotate feature of TFS will fail on the many changes due to the large size of the xml file.

Storing the CRM Solution as Multiple Unpacked Files

- This method results in the Source Control System becoming the ultimate master for all components including source code and CRM configuration/customization. Developers need to take particular care with check-ins and check-outs and ensure that only the specific configuration/customization elements required are updated. The build master should be responsible to track the changes inside the solution folder structure and check for any inconsistencies/deviations.
- Parallel versions of complex CRM solutions can be supported by tracking the individual configuration/customization changes throughout the CRM solution package. The branch and merge operations can be executed in the TFS environment for individual customization elements.

  Note: Direct editing of some portions of the CRM customizations files is supported. For more information, in the Dynamics CRM 2011 SDK, see the topic When to Edit the Customizations File.
- Multiple parallel development branches can be supported using this method for long-term phases. Tracking the individual changes of the customization tree will be possible and many of the necessary merge operations can be executed within the TFS environment. The required manual CRM merge operations will be also supported by TFS merge, producing the list of specific changes between the branches enabling identification of any items requiring manual merge interaction.
- Supporting multi-year projects is significantly simplified by following the unpacked solution files method. Individual configuration/customization changes can be tracked back to the specific change-set, task, requirement, or bug. The TFS environment provides the necessary tracking and reporting toolset to manage the entire lifecycle of specific CRM customizations down to an attribute or relationship level.
- When using this method, the daily build process is responsible for integrating all of the configuration/customization elements into a single solution file which is achieved through automation of the pack operation. This includes any assemblies, web resources, and plug-in registration steps required as part of the final CRM solution. The greatest benefit of this method is that there is no need for a running CRM instance to create the solution package. TFS is the master repository and the build process is essentially an offline build. This simplifies scale out of the build agents and the number of different build definitions that can be supported.
- Consistent versioning of all solution elements can be realized with a simple processing step, executed as part of the pack operation, to replace the fully qualified name of the assemblies with the current build numbers. Assemblies and web resources can be copied directly into the CRM solution package.
- Using version control and unpacked files, all TFS features for change-set and work item tracking are available to use. The TFS file and row-based tracking can be used to track the lifecycle of any CRM configuration/customization changes. The individual files for entities, relationships, forms, workflows, and templates can be tracked back to creation and to all modifications. Specific modifications will be bound to smaller, distinct check-ins, which can be traced back to developer work-items and requirements.

- This method provides opportunity to implement consistent quality gates which can simplify management and monitoring and improve quality assurance. The code churn of a Dynamics CRM solution will be measurable. Individual check-ins and changes can be validated and analyzed from a quality and risk perspective, and the ability to undo specific, unwanted changes or check-ins is provided.
- Tracking the history of specific changes is supported by the standard View History feature of TFS at a more granular level of each individual configuration/customization element. The Annotate feature will also work since the files will be smaller, more specific xml fragments. For example, this approach makes it possible to get the entire history of a relationship definition: who created it, when, when it was changed to cascade all, then back to cascade none, etc. The built-in links between work-items and change-sets will also provide the ability to trace changes to specific bugs, reported by a specific business user.

Managed Versus Unmanaged Solutions

There are two distinct aspects of unmanaged configuration/customization handling in CRM 2011: the unmanaged layer and the exported unmanaged solution.

The unmanaged layer contains all configuration and customization changes that are not part of a managed solution, either created or upgraded within the CRM organization or imported through an unmanaged solution. The unmanaged layer has a different behavior than managed solution layers regarding visibility and merge behavior. More information about solution layers can be found in Appendix D: CRM Solution Concepts.

An unmanaged solution *file* is independent of the specific layering configuration present within a CRM organization. The export of an unmanaged solution will contain the copy of all included elements, which are visible *as a result* of the system solution layer configuration, but will not include any information about the specific layer configuration or dependencies. An unmanaged solution always contains full customization elements – it does not contain any difference information (except for RibbonDiffXML, which is persisted as a delta).

An unmanaged solution can be exported-reimported-edited without constraints in any CRM environment.

Managed solutions behave differently as they can only be exported from one CRM instance, the one containing the unmanaged components. The exported managed solution can be imported into any other CRM organization (containing the required dependencies, if any) but cannot be edited directly or exported from there. Elements marked as customizable may be edited, but it will result in an unmanaged layer customization.

A managed solution file contains the same full customization definition for most of the elements as that of the unmanaged solution file, except for a few elements which only propagate the difference/delta:

- FormXML
- SiteMapXML
- RibbonDiffXML (same persistence as in unmanaged solutions)

Versioning the unmanaged solutions using the unpacked solution files approach typically satisfies enterprise project versioning requirements.

Note: For additional details about internal storage and logic of managed solutions, see the blog posting Internal storage of SiteMap and Form customization in managed solution packages.

Scenarios

In enterprise deployments, there are two completely different aspects of version control processes:

- Source code control through development
- Release management versioning to production environments

The previous sections covered the source code control aspects, specifically how to manage the process in the development phase, including work-item tracking and source versioning. The following scenarios illustrate and reinforce the points made above.

In customer scenarios, typically the keyword for environment versioning is *change*. The most important questions to ask and answer are the following:

- Are there any change management processes at the customer? Which scenarios require support from the CRM solution perspective?
- Is there a need to be able to redeploy the application with a single CRM solution – starting with an empty environment?
- Is there a need to be able to deploy different configurations of CRM solutions for specific organizations? (for example, Base+Feature1, Base+Feature2, Base+Feature1+Feature2)
- Is there a need to support both hotfix packages and single-solution deployments? (for example, Update Rollups, creating a new organization instance later – without the need to apply layered hotfixes)
- Is there a need to support, roll-out and maintain environments with different hotfix versions in parallel?

You should thoroughly consider the versioning requirements for the application at the beginning of the development phase and plan the versioning and hotfixing solution structure that will be able to support it.

Supporting One CRM Solution for One Production Deployment

In the simplest scenario a single CRM solution will be created for Contoso. The solution needs to be deployed with a single roll-out to one production environment.

The application functionality contained within the single CRM solution may be configuration changes and/or may contain a degree of customization including web resources, plug-ins or workflows.

In a small development team or single developer scenario, storing the binary zip solution file under source control may be leveraged as the approach to store CRM solutions. In larger, multi-developer or multi-team scenarios, the unpacked solution management approach provides more benefits around flexibility and versioning, even when considering a single CRM solution scenario.

The base source structure can be created at the beginning of the project with an unmanaged solution (as illustrated in the previous sections).

Irrespective of the development methodology, for example agile versus waterfall, a version of the development to-date can always produce a managed solution for deployment from the automated daily build process. At certain points, such as at the end of a sprint or whenever dropping a release into a model office or acceptance environment, the managed solution that has been deployed should also be maintained under version control. An additional approach is to branch the release at this point to be able to recreate the build if necessary but also to hotfix it.

During the acceptance test phase of a release, one or more hotfix packages may be required. Typically, any existing data within the environment needs to be preserved not only to continue testing but also due to constraints around the time required to re-provision the environment. This can be supported by minimal, hotfix solutions as detailed in the Hotfix Process section.

Within the acceptance phase, hotfix solutions are typically cumulative. Developers work on a branch for the deployed release and, in an appropriate window, can create the hotfix solution through the daily build process and deploy onto the acceptance environment. Though work is conducted on a branch, developers make the necessary changes within a version of the Visual Studio solution for the CRM solution containing the full unpacked customization package.

If there is no requirement to preserve the data within the acceptance environment, the recommended deployment strategy is to "reset" the environment to the previous known state which may have no data or a pre-defined baseline of data (for example, reference and customer data) and install the full managed solution again. The reset of the environment may be achieved through a number of approaches including database restore or snapshot reversion if using virtualization technology for example.

Upon completion of the acceptance phase, any hotfixes developed on the branch within Source Control should be integrated onto the trunk. A new branch for the production release should be created from the trunk and a full managed solution package should be built and deployed to the pre-production or staging environment. If it is intended for the UAT environment to assume this role, it should be "reset" and have the new package deployed. This tests and provides confidence that the deployment approach will work for production. It also ensures that exactly the same codebase is on the pre-production and production environments which supports and adds confidence to the process of problem diagnosis and subsequent deployment of hotfixes. This approach also ensures that development of the next version may start without affecting deployment, go-live, and post go-live support of the current version.

After go-live, it is recommended to take a backup of the production environment and restore it onto the pre-production environment. This provides a consistent, reproducible test environment for any production issues. Sometimes this is not possible for data privacy reasons that may necessitate de-sensitizing the data first or removing certain types of data (for example, financial statements).

During development of version 2 of the application, the recommended approach for daily testing is to start from a baseline that is consistent with what has been deployed to production (in this example, the baseline version 1 deployment plus hotfixes). This would be stored as a snapshot of the environment that can be restored each day before deploying the current day's cut of the managed CRM solution. This practice ensures the deployment approach that will be used for UAT, pre-production, and ultimately production is tested early and often.

Having a strategy for Update Rollups is essential to live operation of the CRM application. If virtualization technology is being utilized, it is important to keep the snapshots in-line with the current state of the production environment. This requires any change in Update Rollup level within the production environment to also be applied and committed to the snapshots within the test and pre-production environments.

Feasibly, there may be multiple hotfix solutions for version 1 in parallel to version 2 development. There are three options for integrating the changes from version 1 hotfixes into the version 2 development branch:

- No merge:
  - Install Version 2 on top of the hotfix solution layers.
  - Export the managed solution containing dependencies and differences to all of them.
  - Preserve hotfix layers after installing v2.
- Manual merge:
  - Install the unmanaged hotfix solutions before v2 and include all hotfixed elements in v2 solution.
  - Export v2 solution including all v1 hotfix customization and check-in under TFS.
  - The managed hotfix layers can be removed after installing v2 and before exporting the managed v2 solution, resulting in dependency and differences only to v1.
- Merge using TFS unpacked structure (recommended):
  - Similar to the previous option, include all hotfixed elements in the unmanaged v2 solution, and check-in under TFS.
  - Execute TFS merge operation between the v1 (hotfix) and v2 (main) branch to preserve hotfix history

The different aspects of deploying hotfix and next-version solutions are detailed in the Hotfix Process section.

Supporting One CRM Solution for Multiple Regional Deployments

The second scenario is quite similar to the single deployment. As described in the Scenario, the core solution will be reused by multiple business groups through a regional deployment model. In the simplest case, this approach is no different from supporting one CRM solution for one production deployment, merely requiring repetition of the deployment steps on each CRM deployment. In this case, all regional environments are versioned and patched together to the same version of the CRM solution.

A more complex scenario can occur if the regional deployments have a different version of the core solution or, over time, this situation arises. This would result in a need to support multiple, parallel versions of the core solution for development, acceptance, and deployment. These environments may also have different UR levels, resulting in more complex scenarios.

For supporting multiple parallel versions of a single core solution, it is recommended to use unpacked solution files in conjunction with the branch and merge techniques covered in the previous scenario to augment the approaches covered purely from the vantage point described in the Lifecycle from a CRM Solution Perspective section. The source tree may support all scenarios by creating specific branches for the required combinations – Update Rollup, version 1, and hotfix versions.

For simplified management of the complex scenario, you can leverage the TFS release->SP branching model. It provides one main development branch and multiple release branches (one for each specific system hotfix level).

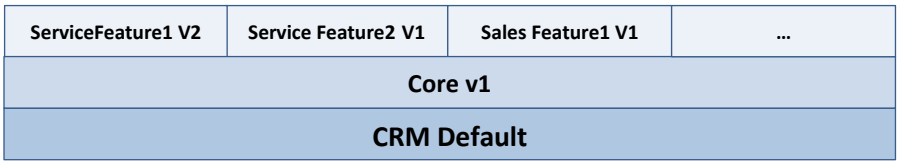
The hotfixes specific to an environment should be created on a development environment provisioned with the branch build. The hotfix solution and release of this build can be created from the branch. The actual hotfix changes may be irrelevant to other environments, or it can be merged back to the main branch – or even directly to other branches – using the aforementioned merge techniques.

The number of required branches typically depends on two dimensions; the number of different UR levels and the number of hotfix versions. The recommended approach is to keep all environments for a release at the same UR level; this way it is only required to support the different hotfix levels with branches.

Supporting and Managing a Multi-Solution Environment

The most complex scenario also includes management of multiple solutions and layers. In these cases, the number of combinations to support can explode, as a multiplication of the number of solutions, number of versions and hotfixes, and possible combination of solutions (not mentioning UR levels).

The recommended approach is to keep the number of solutions low wherever possible to avoid complexity and to ensure the solution customizations are independent from each other. This will enable the techniques for managing multiple versions described in the previous section to be extended and applied for managing multiple solutions.



From a version control perspective, the management of CRM solutions for complex scenarios is best supported by the unpacked solution structure. The creation of specific branches, specific managed solutions, and the merge operations still requires manual work, but it will be supported by the change tracking of TFS; the build master will always know which elements changed compared to which branch (which environment), making decisions possible in terms of what to include and what to merge.

Using versioning via the full XML solution file makes it difficult to track changes in the solutions and branches of the application, and it will provide no information regarding the differences between the branches. The management and administration of the solutions must be executed manually, resulting in a lot of extra effort and a higher risk of errors.

Editors for Component Sources

Dynamics CRM as an Editor

The Dynamics CRM out-of-the-box editor remains the primary editor for CRM entities and forms, and with full WYSIWG support, it is very effective for these purposes.

Note: The customization UI is only accessible using the base language. Both the base customizations and the language translations should be managed during a development lifecycle (as separate or bound work-items).

The Dynamics CRM administration UI serves as the editor for templates, security settings, language translations, and for organization side configurations.

Specific Editors

The Dynamics CRM customization XML file supports direct editing of specific elements as XML. The editing of XML content may be supported by external tools for example the Ribbon Editor, SiteMap Editor, etc.

Note: Manually editing the customization.xml is only supported for the following elements:

- Ribbon
- SiteMap
- ISV.Config
- FormXML
- Saved queries
- Chart definitions

Defining any other solution components by editing the exported customizations.xml file is officially not supported; in the Dynamics CRM 2011 SDK, see the topic See When to Edit the Customizations File. For information about supported and unsupported customizations, in the Dynamics CRM 2011 SDK, see the topic Supported Extensions for Microsoft Dynamics CRM.

- **Visual Studio Editor**. The Visual Studio IDE is the primary environment for developing custom assemblies and components. Use the CRM Developer Toolkit to support creating the custom development artifacts of a Dynamics CRM solution.
- **Text Editors, Excel**. Text resources, data files and import/export files may be edited through any text editor or through the Visual Studio IDE. The recommended approach to import data into Dynamics CRM is to use the XML spreadsheet format, which is easily editable through Microsoft Excel.
- **3rd Party Editors**. For editing specific web resources or external components, other editors may be required. Using TFS enables the necessary support to connect 3rd-party developer tools to the source control via its plug-in interfaces (for example, Java or other IDE).

Branch/Merge Strategies

The following situations likely will be faced and must be managed for long-term projects that span multiple phases:

- Separate releases of feature packs overlapping in development requires branch/merge and parallel environment versioning
- Hotfixes and separate management of next-phase development and live support
- Different teams managed by different vendors who are developing application features and hotfixes
- Changes during an upgrade project, including supporting changes for the old environment
- Update Roll-up version regression testing and roll-out scheduling parallel to the main development branch

During a long project the recommended way is to remain with the single-team, single-release branching strategy as far as possible to reduce complexity. Some situations result in more complex scenarios, making it necessary to fall back to multi-team, multi-feature branching strategies as described above.

Using the unpacked and source-controlled CRM customization tree, the customization merges are executed offline (without the need for CRM). Ideally, it is recommended to isolate features into separate CRM solutions so that no merge conflicts need to be resolved within CRM customizations.

Build

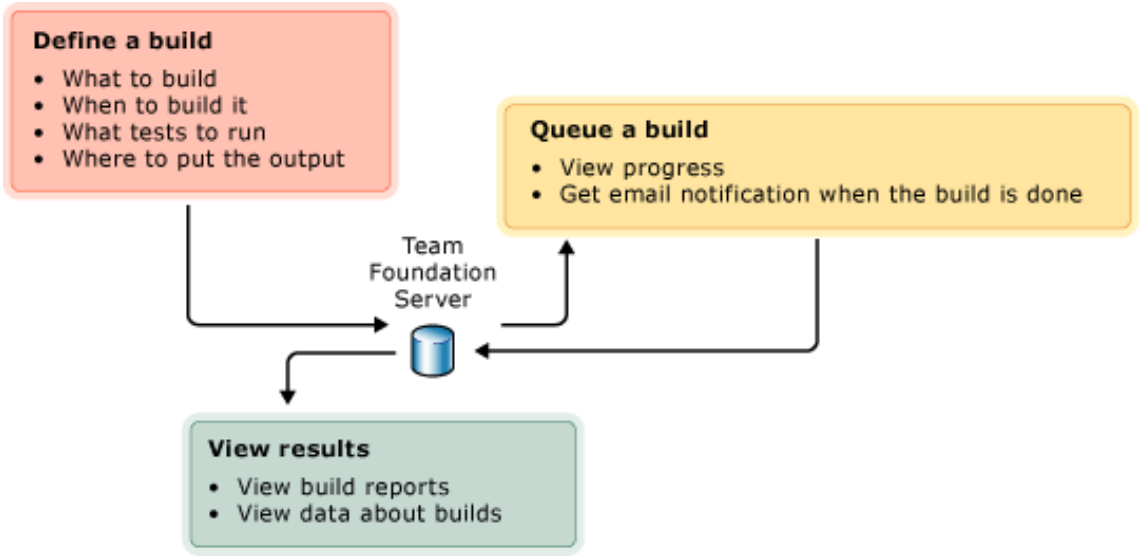
Builds are executed as a process of combining all the components of the solution in a repeatable manner with the main objective being the reduction of errors and minimization of the time to deliver software changes to a production system. An important element of that is a series of quality checks as part of the build process that are achieved by validation checks beginning in the earliest phase possible (that is, on the developer's environment before checking-in changes to Team Foundation Server). Additionally, validations can be executed during the check-in process to Team Foundation Server and after checking in to the Team Foundation Server.

The basic intention of quality checking within builds is to detect errors as soon as possible before any other team member or any other system will be affected by an error, minimizing the impact of that error and the cost and time to resolve it. Build execution itself, in general, is either a manual or an automatic process.

With Team Foundation Build, you can create build definitions to automate compiling applications, running associated tests, performing code analysis, releasing continuous builds, and publishing build reports.

To build an application, you create a build definition to specify what projects to build, what triggers a build to run, what automated tests to run, and where to deploy the output. This information is stored in TFS from where it is retrieved when a build runs. After the build runs, data about the build results is stored within TFS and is available to view through build reports.

The following illustration shows the three main phases of building an application:

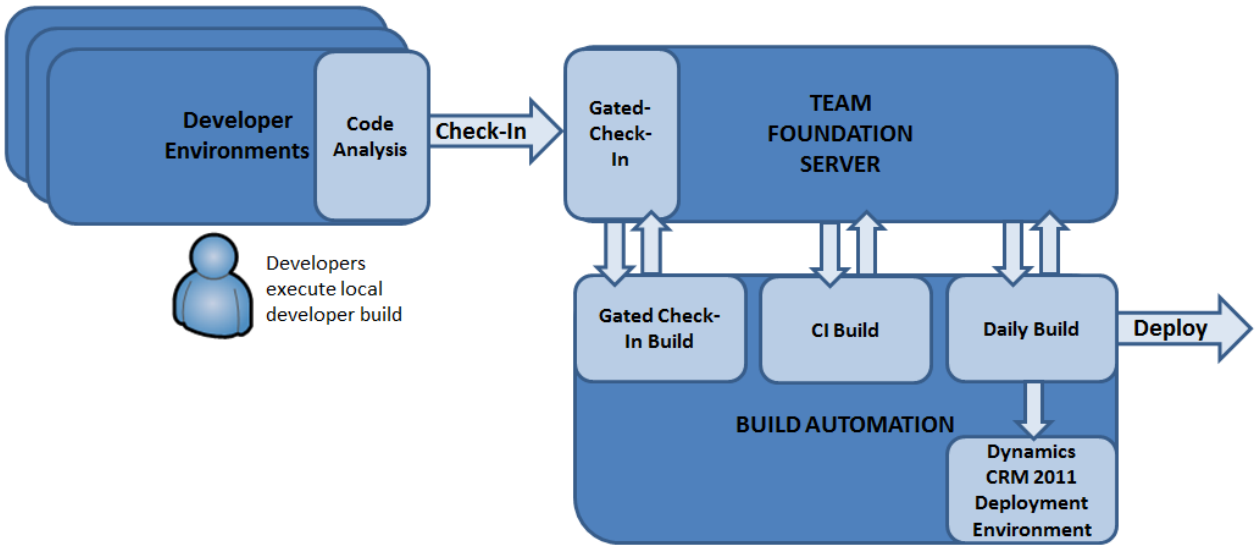


An automated build process compiles, deploys, and then runs build verification tests (BVTs) against the latest source code for a project at regular, predetermined intervals. Then a "build report," detailing the success or failure of the build process, is disseminated to the project stakeholders. The build report is analyzed to determine what areas of the project need attention and/or if the project should be rolled back to an earlier version/build.

The power of an automated build process is that it can be scheduled to execute out-of-hours, which can help ensure the stability of the project without taking cycles directly away from the development time. This topic provides an overview of the Dynamics CRM build process alternatives and describes how build verification testing fits into the build process.

Note: For information about TFS build process automation concepts, on MSDN, see Building the Application.

The following diagram represents the concepts and topics covered within the following sections.



Local Developer Build and Development Workflow

The local developer build is the first gate in terms of build quality checks. The local developer build is part of the usual developer's work and is included in the steps a developer takes when developing based on a specific TFS work-item.

It is performed to ensure the developer's changes work in their developer environment when combined with the latest version of all "source" files from source control and to ensure that later automatic or manual builds will not break and no blocking of downstream project processes occurs.

The local developer build is a manual, on-demand process that, when executed, builds and deploys all affected components to the developer's isolated developer environment followed by executing the automated unit tests that have been defined for execution.

The local developer build is executed when code changes regarding one or more work-items have been implemented. It is executed before checking-in to Team Foundation Server and typically runs in "release" mode.

An ideal development workflow from a developer perspective would be:

- Reset the developer environment CRM instance to a known state.
- Accept work item in Team Foundation Server.
- Get the latest version of source from the appropriate TFS branch to work on.
- Execute Local Developer Build/Deploy in local developer environment.
- Implement work item.
- Execute Local Developer Build/Deploy in local developer environment.
- Test in local developer environment.
- Get latest version from TFS.
- Resolve potential conflicts in workspace.
- Execute Local Developer Build/Deploy in local developer environment.
- Retest in local developer environment.
- Check-In change-set and verify Continuous Integration build result.

Check-in Policies and Gated Builds

Quality gates in the form of check-in policies can be specified that need to be met prior to successful check-in of a change set. A number of these are available out-of-the-box and many more are available from CodePlex and various other community sites.

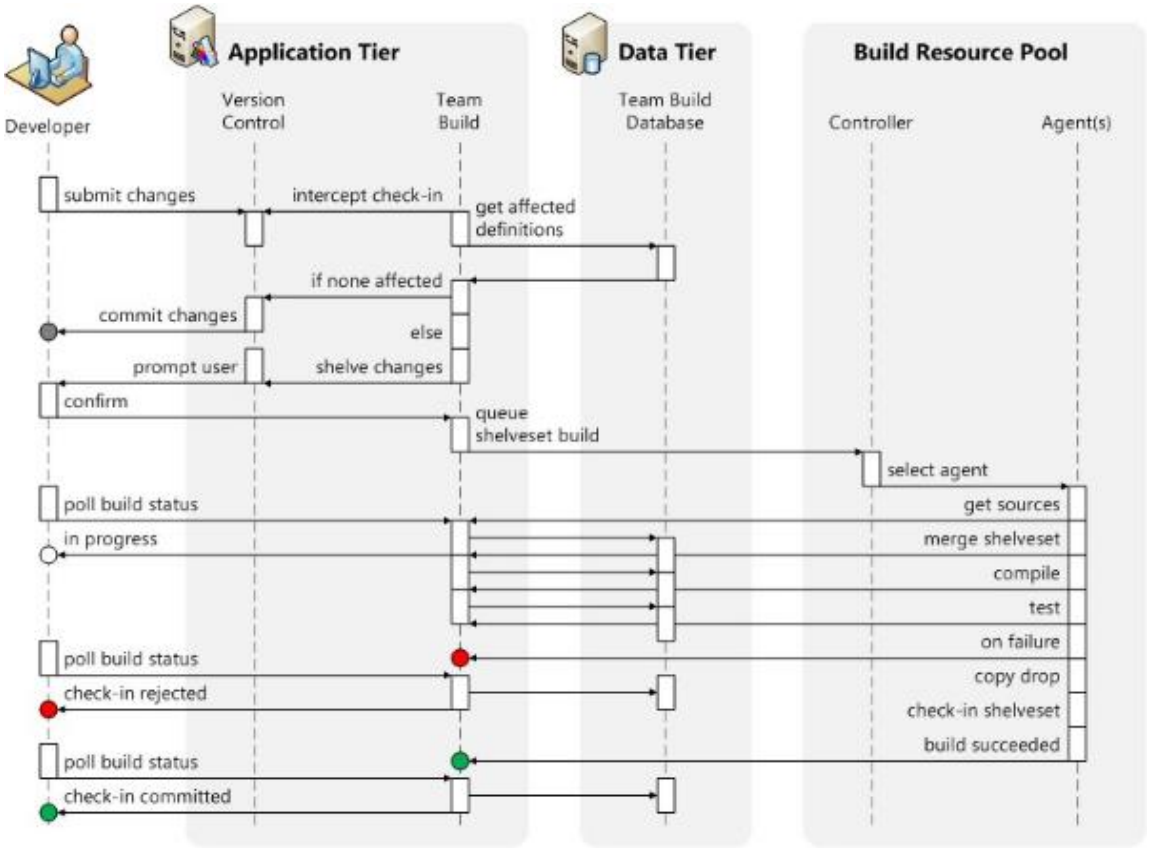
The Visual Studio code analysis policy is one such example and only accepts change sets for check-in where the code complies with the rule sets defined for the Visual Studio project. This helps to ensure a level of code quality early in the development phase and provides better maintainability of code in general.

When a developer checks-in changes that break the build, the result can be a significant hassle for small teams. The cost to larger teams can be expensive as measured by lost productivity and schedule delays. You can protect some or all of your codebase against this problem by creating a gated check-in build definition.

When a gated check-in build is created, changes that the developer submits are placed in a shelve-set and automatically built in your build system. The build must be successful for the check-in process to be completed.

Note: For more information, on MSDN, see Check in to a Folder that is Controlled by a Gated Check-in Build Process.

To use the gated check-in build, a build automation process has to be implemented. The gated check-in build flow is shown in the following diagram:



- For available built-in rule sets for Microsoft Visual Studio Code Analysis, on MSDN, see Code analysis rule set reference.
- A configuration sample for the Gated Check-In can be found in the MSDN blog posting Preventing check-ins to TFS that contain code analysis warnings.
- For details about configuration of a Gated Check-In build process to validate changes, on MSDN, see Define a Gated Check-In Build Process to Validate Changes.

Continuous Integration Build

In general, a developer check-in operation cannot be considered successful until the continuous integration build completes successfully. Continuous integration (CI) builds are builds that are executed regularly – typically on each change set.

The CI process triggers event-based builds on a build agent. An event starting a CI build process may be the check-in event raised by a developer checking-in a change set to Team Foundation Server.

It is possible to submit a change set for check-in without actually directly checking-in the files. The CI build process can take the change set as a "shelve-set," together with the most recent versions of all other files not contained within the change set, compile them, and run any other gates prior to allowing the change set to be checked-in to the main source control folder structure. If the CI gates are passed and the code is merged and committed into the main source control structure, the changes are available to all developers and will also be included in the next daily build.

Typical gates are:

- Code reviewer specified (someone who has been identified as having reviewed the code for correct functionality and at least ensuring that it compiles)
- Style and code analysis
- Unit tests pass results

Note: See the Test section later in this document for additional information on testing and test automation.

Within the CI build, it is likely that the version number on assemblies will be incremented and optionally the filename of the CRM solution to be exported will also contain this information. With CI builds there is near-term feedback on every check-in related issue and blind spots are eliminated. The early detection of issues breaking the daily builds or any other build process reduces the risk for regression.

Enriching team awareness with these build results fosters a team-based culture where it is important to resolve broken CI builds as quickly as possible. To support the process, an automated "build break" e-mail notification can be set up. For additional information, on MSDN, see Run and Monitor Builds.

Daily Build

Daily builds are scheduled and automated build processes run on the latest source version stored within Team Foundation Server. Every operation that needs to be performed during the build process is defined within the build definition. Daily builds ensure that a complete version of the project can be successfully built and deployed.

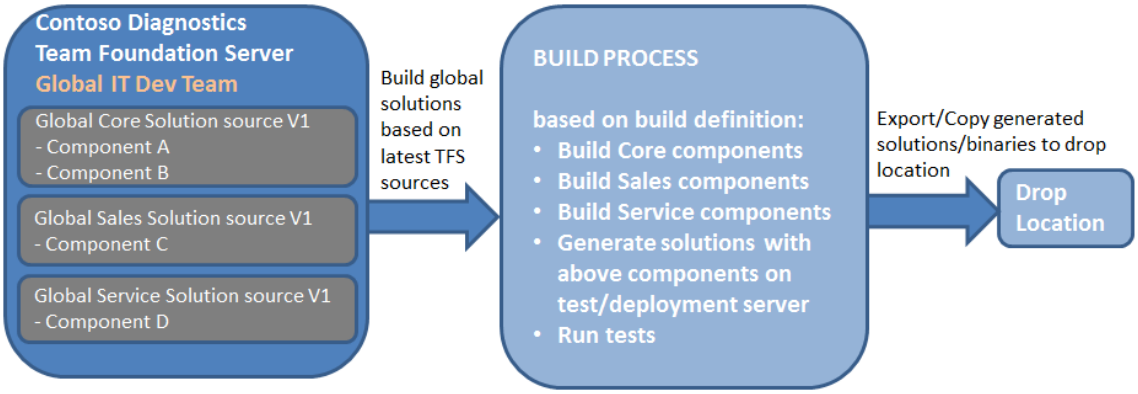
There are two variants of daily builds:

- **Online**: Requires a CRM server to produce a CRM solution as an output
- **Offline**: Does not require a CRM server; TFS is the master repository and the SolutionPackager tool is used to bring the components together into a CRM solution package.

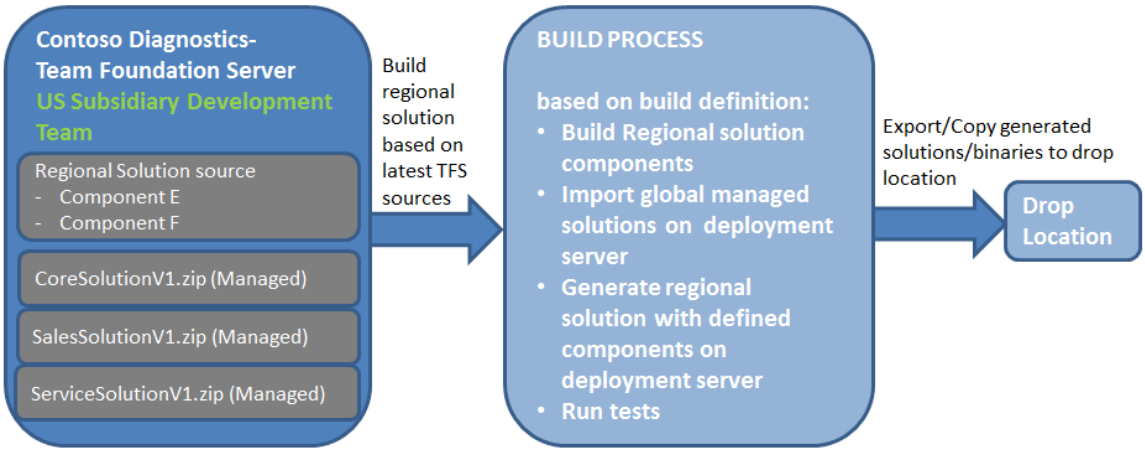
Irrespective of which approach is used for creation of the CRM solution package or "drop," if there is a requirement to perform a level of verification testing (for example, smoke testing or build verification testing) prior to passing the build to the test team, a running CRM instance will be required.

A typical approach to this situation would be to have a virtual instance of Dynamics CRM that can be scripted to be reset to a known state and started up to receive the latest build. Unit tests and smoke tests can be executed and code coverage metrics can be captured for this deployment. This approach can be further extended to Lab Management approaches as explained in the Environment Automation section.

From the perspective of the Scenario, the Global IT development team works on the centrally developed core solution and, additionally, on the Global Sales and Service solutions respectively. These solutions are developed in adherence with the development processes shown in the previous sections of this paper, and the CRM solution packages are generated in an appropriate daily build process.



The regional development team in the US subsidiary of Contoso Health builds a regional solution on top of the global solutions released by the global development team to extend and adapt the features to support regional regulatory requirements and processes. The dependent solutions have to be considered in the build definition.



Working with Developer Environments

There are several options for setting up a developer environment and, as such, only general recommendations can be given. The decision on which development environment topology to be used has to be made on a project-by-project basis and can often be constrained by limiting factors such as hardware availability and policy when working on a customer site. For decision guidance see the development environment function/cost-related pros and cons in Appendix F – Development Environments.

Basic differentiation can be made between working in an isolated environment with a full Dynamics CRM/SQL Server installation or in a shared environment with Visual Studio on a local developer machine connected to a shared CRM environment. These options are considered in the following sections.

Using an Isolated CRM Instance per Developer Connected to TFS Centrally

Setting up an isolated developer environment connected to a central Team Foundation Server requires performing the following steps:

1. Set up the software environment
2. Restore the Dynamics CRM 2011 database to a known state (optional)
3. Import project specific Dynamics CRM reference data (optional)
4. Connect to Team Foundation Server and get the latest version of files from source control
5. Build and deploy

The following sections provide additional detail about each of these steps.

Set up the Software Environment

Typically there are a number of platform and application software installations required, for example:

- Windows Server 2008 R2 Standard Edition/64, Active Directory Domain Services
- Microsoft SQL Server 2008 R2 Standard/Developer Edition
- Microsoft SQL Server 2008 Reporting Services
- Microsoft Visual Studio 2010 Premium/Ultimate
- TFS Power Tools for Visual Studio 2010
- Dynamics CRM 2011 server and Update Rollup Package and related Dynamics CRM components such as:
  - Dynamics CRM 2011 language packs
  - Dynamics CRM 2011 SDK

  Note: Details for installations of Dynamics CRM and related components can be found at: Microsoft Dynamics CRM 2011 Installing Guide
- Supporting developer tools such as:
  - MCS Metadata and Administration Tool
  - Dynamics CRM SiteMap/Ribbon Editor
  - Developer Toolkit
  - JSLint, LinqPad, Odata Query Designer
  - Script#, Fiddler, Diagnostics Tool, Resharper

Note: In a development environment it is normal to cover software licensing costs for Microsoft software applications through the MSDN licensing model.

While developers can install these locally for themselves, this does not ensure that everyone is using the same tool versions over time. The recommended approach is to automate the task of deploying these applications by running a command line script copying them from a project file share or by storing them in a folder on Team Foundation Server. This process ensures all developers work on the same tool set with the correct software versions and appropriate licensing. The developer then installs tools or imports into Dynamics CRM 2011 if required.

Restore the Dynamics CRM 2011 Database to a Known State (optional)

If a defined Dynamics CRM 2011 database backup is available this database backup can be restored and the organization can be setup/imported via the Dynamics CRM 2011 deployment manager.

Import project-specific Dynamics CRM Reference Data (optional)

Import of project specific CRM data such as users, teams, business units, subject tree, etc. can be achieved with the import wizard/automated import job.

Connect to central Microsoft Team Foundation Server and Get Latest version

The next step before the developer can start working is to set up a connection to the appropriate Team Foundation Server where the project is managed. In advance, the team project administrator has to set up the permissions for the team project within TFS for the developer accordingly. The developer then connects to the team project and gets the latest version of files from source control. This process should also be automated via a script.

Build and Deploy

The remaining step for the developer is to build and deploy the project solution components to the development environment.

Using a Shared CRM Instance for All Developers Connected to TFS Centrally

Setting up a local developer client connected to a central CRM instance and Team Foundation Server requires performing the following steps:

1. Set up the software environment
2. Create/restore the CRM organization
3. Import project specific Dynamics CRM reference data (optional)
4. Connect to Team Foundation Server and get the latest version of files from source control
5. Build and deploy

Set up the Software Environment

This process is no different from that described in the section Using an Isolated CRM Instance per Developer Connected to TFS Centrally, but the applications and platform requirements are more limited, including for example:

- Microsoft Visual Studio 2010 Premium/Ultimate
- TFS Power Tools for Visual Studio 2010
- The need to support developer tools as described in the section Using an Isolated CRM Instance per Developer Connected to TFS Centrally

Once again, the recommended approach is to automate the process of installation via a command line script (or similar mechanism) to streamline the process and ensure consistency across all developer environments.

Create/restore the Dynamics CRM 2011 Organization

Even when using a shared CRM Instance, it is possible to achieve a level of developer isolation by using multiple CRM organizations. If a defined Dynamics CRM 2011 database backup is available, this database backup can be restored and imported into the shared CRM deployment as a new organization for the developer. Alternatively, a new development organization could be created for the developer.

Import project-specific Dynamics CRM reference data (optional)

Import of project-specific CRM data such as users, teams, business units, subject tree, etc. can be achieved with the import wizard/automated import job.

Connect to central Microsoft Team Foundation Server and Get Latest version

The next step before the developer can start working is to set up a connection to the appropriate Team Foundation Server though which the project is managed. In advance, the team project administrator has to set up the permissions for the team project within TFS for the developer accordingly. The developer then connects to the team project and gets the latest version of files from source control. This process should also be automated via a script.

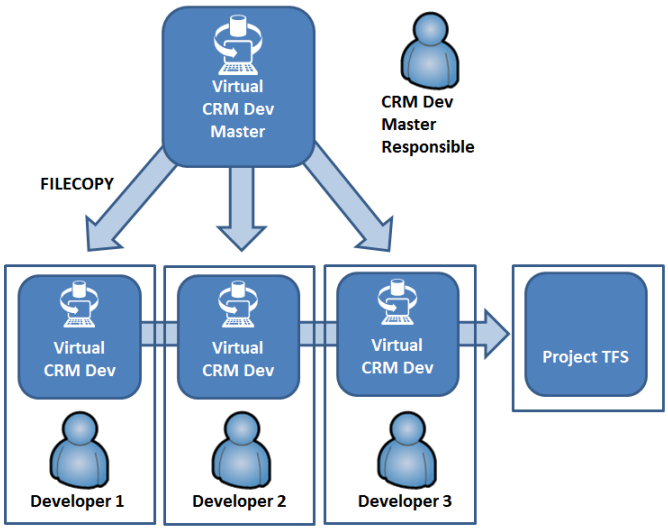
Build and Deploy

The remaining step for the developer is to build and deploy the project solution components on the development environment.

Enterprise Development Scenario with a Centrally Prepared Development Environment

For enterprise development scenarios with several developers using a centrally prepared CRM developer environment, using a CRM Developer Master. In effect, this master environment is an all-in-one (CRM in a box) virtual environment containing Active Directory, Dynamics CRM 2011 Server, and Microsoft SQL Server based on a Windows Server platform that provides developers a totally isolated environment in which to work.

The prepared developer environment can be centrally administered and provided to the development team. As with the previous topologies, this set up ensures that each developer works in an environment that has the appropriate software versions and the toolset exactly as specified for the project.



Developers who are new to the team or who need to refresh their environment can copy the master virtual image from a central project location to their local machines or have it hosted and provisioned on a centrally hosted server.

Note: If multiple copies of the virtual image are in use, care should be taken to use an internal or private virtual network to avoid collision. If these instances do need to be exposed on the network, it is advised to prepare the image accordingly: for example, through the System Preparation Tool (sysprep).

The developer starts working by initially connecting to the projects TFS server, getting the latest version/branch of the solutions the developer needs to be working on and building and deploying on the local environment. If available, project-related components can be pulled and deployed from the drop location of the last daily build.

Capturing Known States/Reverting to Known State

Capturing states of a development environment in a development process within a project's timeline is necessary to be able to "reset" to a certain state at a later point of time. In the scenario above, it may also be necessary for the IT department at Contoso to be able to "reset" an environment to the state that is currently in production in order to create a hotfix for the production environment. Capturing states of a Dynamics CRM development environment is possible on different levels, as described in the following sections.

Microsoft SQL Server / Dynamics CRM Organization level

Through Microsoft SQL Server database mechanisms, it is possible to capture the state on a database level by backing up the organization database (OrganizationName_MSCRM) and the configuration database (MSCRM_CONFIG), and then restoring tem to the same computer or detaching and attaching database files. These steps should be automated via a script.

Checking-in the “known state” database files to Source Control is recommended. In addition, you can transfer to another development environment, restore, and import of organization within Dynamics CRM Deployment Manager.

Note: For additional information, in the Dynamics CRM Implementation Guide, see the following sections:

- Backing up SQL Server, including Reporting Services
- Import a Microsoft Dynamics CRM 2011 Organization

Use of Virtual Machine Snapshot mechanisms

Using virtualization technology such as Hyper-V, a virtual machine snapshot may be taken at any time. The advantage of virtual machine snapshots is that the complete environment state is captured rather than only the Dynamics CRM database. In this case environment changes like Dynamics CRM Update Rollup level are also fully considered.

In the scenario at Contoso Health, the IT department may take a snapshot of a developer’s virtual machine as the project goes live in production to store the environment used for development of the production hotfix solutions.

However, when working with virtual machine snapshots, storage requirements have to be considered since snapshots may require a significant amount of hard disk space. For additional information, on MSDN, see Hyper-V Virtual Machine Snapshots: FAQ.

In addition to the above, building and deploying the latest source control content from the appropriate source control branch may be necessary to complete the process of reverting to a certain known state.

Resynchronizing with the Development Team

For Contoso Health developers, resynchronizing with the development team after each work item they complete is necessary to ensure that the code base on their environment (including the CRM Configuration) is synchronized with the rest of the development team. This reduces and, in some cases, avoids merge conflicts at check-in.

The following steps need to be taken to resynchronize a developer with the development team:

1. Revert to known state or reset the developer environment by setting up of the appropriate organization database or reverting to a virtual machine snapshot (see chapter above).
2. Get latest version from appropriate branch in Team Foundation Server.
3. Build and deploy the Visual Studio solution components.

These steps should all be automated as part of a script for the local developer build.

Test

Identifying defects as early as possible is the least expensive way to ensure software quality. Best practices and tools can help your team to minimize the cost of preventing and fixing defects by maintaining the quality of your project throughout its lifecycle.

Your team can more accurately gauge the quality of your project at any time if you find defects, fix defects, and verify fixes as you go along. By testing often, your team and stakeholders can remain aware of the current state of the code and make informed decisions throughout the project. Ultimately, you should be able to answer the question "Can we release?" and understand the implications for the people who use the software.

For enterprise solution deliveries, the following practices are recommended:

- Create a set of automated unit tests for each source element and for the interface of every major component. Writing unit tests may take a considerable part of team members' time, but this is typically considered an investment against future quality (in terms of driving developer mindset as well as catching errors early) rather than a cost. For general testing information, on MSDN, see Creating Automated Tests.
- Create tests for each user story. These should preferably be automated. For more information, on MSDN, see Creating a Test Plan Using Requirements or User Stories.

- Create check-in policies that remind team members to run unit tests before checking in code. For more information, on MSDN, see Add Check-In Policies.
- Set up a continuous or nightly build that runs the full set of tests.
- Monitor test coverage to ensure that all your code is tested. Aim for coverage of at least 70%. For more information and a sample report, on MSDN, see Testing Gaps Excel Report (Agile).
- Run manual tests near the end of every development phase.

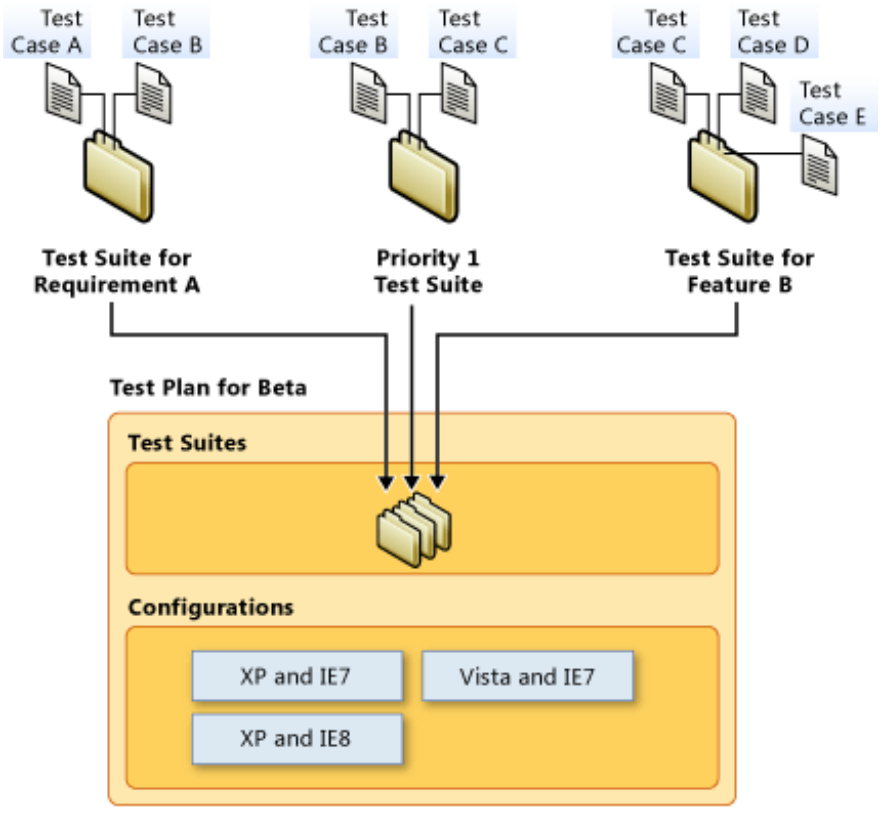
Your team can manage and scale these testing activities early in your project by using the integration between Microsoft Test Manager, Visual Studio ALM, and Visual Studio Team Foundation Server. For more information, in MSDN, see Testing the Application.

General Topics

Test Management

You can use the **Testing Center** in Microsoft Test Manager from Visual Studio 2010 to help you plan your testing effort, based on your approach. You create a test plan to define what you want to test, and then you can measure your testing progress. Test plans can be as simple or as complex as you need, based on your testing approach. You might want to create a test plan for each sprint if you are using an agile methodology. Then you can add the user stories for each sprint to your test plan and create test cases for these user stories. Or, you might create a test plan for each specific milestone if you are using another approach.

You can create test suites in your test plan to group your test cases into suites, based on your needs. You can add a requirement to form a suite in your plan that contains all the test cases that are linked to this requirement. You can assign a set of default configurations to your test plan that you want to cover for quality purposes. You will be able to view which tests have passed or failed for each configuration and how many tests you have left to run. The following illustration shows the key components that are part of your test plan.



Test Automation

You can create several types of automated tests using Visual Studio that enable you to test your application more efficiently. Automated tests run test steps for you and determine whether the test passes or fails. These tests can be run more quickly and more frequently. Automated tests can quickly test whether your application is still working correctly after code changes have been made to your application. Automated tests are created using Visual Studio.

After you create any of the following types of automated tests by using Microsoft Visual Studio 2010, you can run:

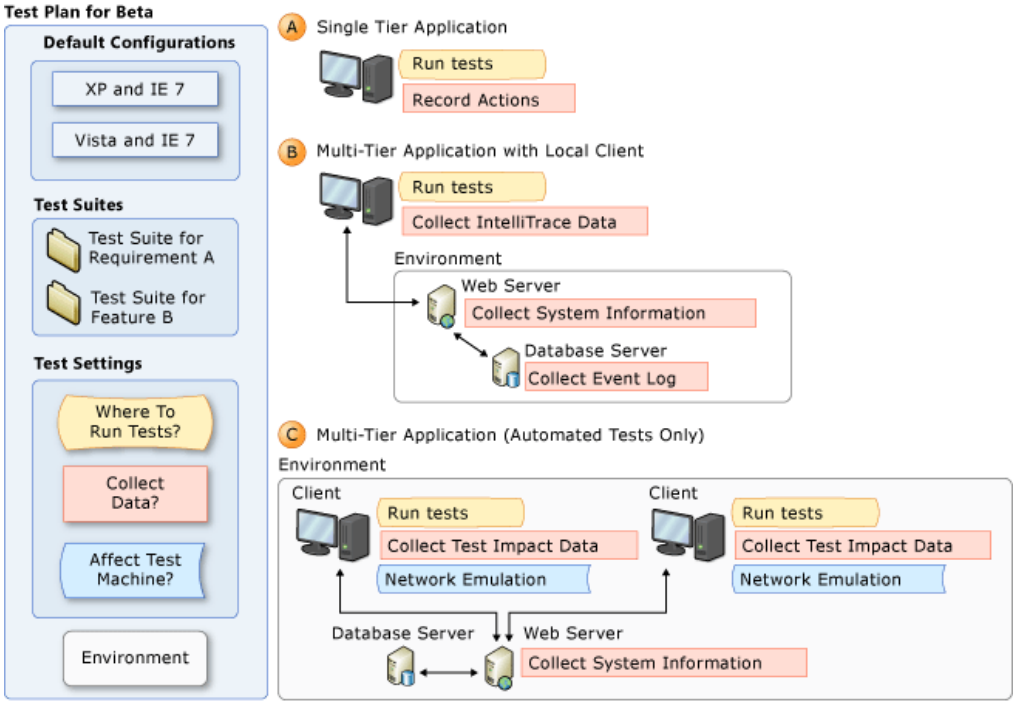
- Unit tests.
- Coded UI tests.
- Database unit tests.
- Load tests.
- Generic tests.

Many methods of running your automated tests are available, depending on how you want to run tests and view the results. If you run your automated tests by using a test plan, you can view your testing progress and easily rerun your tests as required. To run your automated tests by using a test plan, associate your automated tests with test cases and run these test cases by using Microsoft Test Manager. To run your automated tests in this manner, you must create a physical or virtual environment to use when you run your tests. For more information about how to create virtual environments to use to run your tests, see Using a Virtual Lab for Your Application Lifecycle.

The environment enables you to run tests, gather data, or perform system actions on machines for each specific role that you add to the environment. A role specifies the purpose of a machine in the environment. For example, a specific role could be called "CRM Web Front end for Customer Portal." A machine can be a physical computer or a virtual machine. You select which machines to use in an environment for each role.

For example, you could run your tests on one machine and gather system information about a machine that runs the web server for your application. Alternatively, you could run your tests on an environment that uses multiple machines and collect test impact data on those machines. In addition, you can also perform network emulation on the machine that runs the web server for your application.

The following illustration shows three examples of scenarios for how you can set your test settings to run your tests by using environments from Microsoft Test Manager.



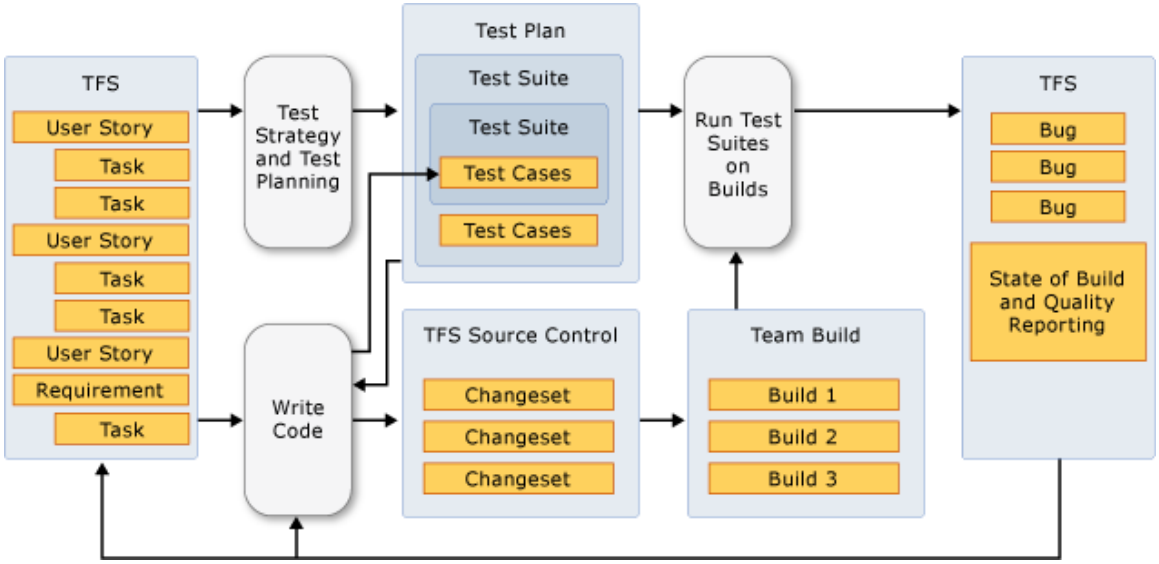
This method of running automated tests lets you view the overall status of any tests in your test plan. If you want, you can view the results of both manual and automated tests together. You can also run these test cases by using associated automation from the command line.

You can also run your tests directly from Microsoft Visual Studio 2010 or the command line without being part of a test plan or without using an environment to run them remotely. In addition, if you add your automated tests to a test category or a test list, your automated tests can be run automatically as part of the build process.

How to Manage the Testing Lifecycle

Testing is an iterative process throughout your project. Refer to the following steps:

1. Make clear testing objectives and make sure that your entire team agrees to them. Based on these objectives, determine your test strategy. For example, your strategy might be "Tests will be run before every check-in, unit tests will have 70% code coverage, and every user story will have at least one automated test."
2. Define your test plan based on project user stories, design assumptions, and nonfunctional requirements in the current sprint.
3. You can add user stories to the backlog and plan them for future sprints. You should match each test plan to at least one sprint and, therefore, should have test cases for all user stories in the sprint.
4. Define and build test cases, such as acceptance tests, unit tests, functional tests, and performance tests.
5. Group test cases into test suites. You can fit these test suites into defined test plans that help guide your testing effort.
6. Run test suites and contained test cases repeatedly throughout a sprint. Start to run tests early in a sprint and continue to add test cases to the test suites. If you want to identify important test conditions and situations, you can apply exploratory testing and have frequent conversations inside your team.
7. Ensure that all the acceptance tests for a user story have passed before you set its status to complete.



Although the workflow can be much more involved depending on the breakdown of software, the previous illustration captures the essence of a workflow among main components.

- Code generates builds.
- Code is influenced by the defined work, test plans, and the quality of builds.
- Test plans, test suites, and test cases are influenced by planned goals and other aspects of the project that this illustration does not show.
- Changes in code can affect test cases.

Software systems have (or should have) a comprehensive set of tests, including unit tests, automated tests, and manual tests. As developers work on the code, the application has to remain under constant scrutiny to ensure it still does what it is supposed to do. That is where testing comes into play. Developers run their unit tests to make sure their code still functions as it is intended, while the QA group tests the application to verify the accuracy, reliability, and performance of the application. However, it is hard to know which tests are actually needed to be re-verified as a result of a code change, so it is common to test the entire area where the code change occurred.

Testing CRM Solutions

Testing scenarios of an enterprise CRM solution usually includes all test types:

- Unit tests
- Database/schema unit tests
- UI, Functional tests (positive, negative)
- Web performance and Load tests
- Generic tests

Unit Tests

Automated unit testing of a CRM solution typically covers the assembly components only. All other components are not easily testable without deploying first (customizations, web resources, workflows, etc.). This is particularly true when considering the offline build/unpacked CRM solution approach since there is no CRM instance to unit test configuration from. Unit testing should be the first quality gate for all development work.

The quality checkpoint for unit testing is coverage or measuring the number of code lines touched at unit test execution. The general recommendation for minimal coverage should be 70%.

Database/Schema Unit Tests

The first step in validation of a deployed CRM solution is the data model and schema. For validating the deployment, you should create automatic tests to validate:

- CRM customizations, entity and attribute existence (using simple fetch queries or DB view select operations).
- CRM views and queries, update operations (using CRM fetch and update operations, checking the results).
- Existence of metadata, data and security elements (by executing queries for returning specific data elements and executing operation on behalf of specific user roles).

Operations can be executed after the successful deployment into a test environment. The test should be automated as part of the build verification test for the solution build.

UI/Functional Tests

Functional tests are important part of CRM solutions. Thanks to the platform's flexibility, the CRM solutions usually have a large set of functionality for a large set of user roles, which can result in a huge number of possible test cases. To be able to maintain solution quality throughout the solution development, it is highly recommended to collect the possible test cases, to prioritize them based on user and functionality impact, and to automate the most important scenarios as part of the build process, using the following methods:

- Scenario based testing using Test Manager
- Automated tests using web or coded UI test
- Executing multi-OS, multi-browser tests
- Executing test with multiple client environment (IE, Outlook, offline client)
- Execute multi-language/localization tests
- Execute ad-hoc manual tests

Note: An important aspect that often is not fully understood is that Dynamics CRM is a platform that has been fully tested. As a result, place emphasis on testing end-to-end business processes implemented on the platform.

Web Performance Tests and Load Tests

Enterprise Dynamics CRM applications are likely to support a large user base and very large dataset. The performance and scalability of the CRM application should always, from the beginning of the project, be considered as a primary quality gate.

The following techniques should be considered for enterprise CRM deliveries:

- Performance POC testing at the beginning of the delivery on a sandbox environment
- Allocating dedicated longer durations for test automations, test runs, and performance tuning during acceptance phase
- Re-running performance tests on gold build, monitoring individual check-ins during acceptance phase regarding performance affects
- Testing client side performance on multiple end-user machines, on multiple bandwidth
- Testing performance on multiple browsers
- Selecting main/most used user scenarios for automated/regular performance testing
- Creating and regularly running automated UI tests
- Including concurrency tests in performance testing (parallel editing, etc.)
- Measuring response time (performance) of automated tests
- Measuring and analyzing manual user test using logs
- Monitoring infrastructure performance at peak loads during load tests and acceptance tests

For more details regarding automated performance testing of CRM deployments please refer to the Microsoft Dynamics CRM 2011 Performance Toolkit section below.

Unit Testing CRM Plug-ins/Workflow Steps

The unit testing of custom components in a CRM solution, without deployment, requires special techniques, since the CRM plug-in context and possible CRM API calls needs to be emulated.

To be able to run a plug-in in a sandboxed unit test environment, you need to create independent, emulated proxies and mocks for the SDK libraries. You can either create your own proxy libraries or use an existing unit-testing framework such as Moles from Microsoft Research.

The Moles Framework

The Moles framework is a Microsoft Research project that aims to provide a systematic, automated approach to unit testing. The Moles framework is a Visual Studio Power Tool and is available as a free download from Visual Studio Gallery; it supports Visual Studio 2010.

The Moles framework actually supports two different kinds of substitution class: stub types and mole types. These two approaches allow you to create substitute classes for code dependencies under different circumstances:

- Stub types provide a lightweight isolation framework that generates fake stub implementations of virtual methods and interfaces for unit testing.
- Mole types use a powerful detouring framework that uses code profiler APIs to intercept calls to dependency classes and redirects the calls to a fake object.

For automatic unit testing of CRM plug-ins, the stub types can be used to emulate the CRM plug-in execution environment.

Note: For more information about the Moles framework, on the Microsoft Research site, see Pex and Moles - Documentation.

Automating CRM Test Cases

The first level of CRM automated test cases consists of the database and schema unit tests that validate the general structure and data of a deployed CRM solution.

The second level of CRM tests should include more extensive API tests to check the consistency of CRM query and update operations and test the functionalities of deployed plug-ins and workflows. Negative and positive test scenarios should also be included to test the error and exception branches of the application.

The third level of automated CRM tests consists of the functional tests. Functional testing of CRM solutions are similar to any ASP.NET or other web applications testing methods, leveraging the web testing framework of Visual Studio environment.

Two possible automation techniques are available using Visual Studio for automated UI tests: Captured web tests and Coded UI tests.

The first scenario is recommended to quickly capture and create automated test cases for a specific CRM form or functionality. The captured test scenario will contain all web requests and responses executed by the client browser. Visual Studio test framework provides a lot of tools to pre- and post- process the tests; however, to be able to rerun a captured Web Test, a lot of manual work is still required (for example, extracting CRM view IDs, entity IDs, parameterizing data entries, and randomizing search clauses and data). The captured and parameterized Web Tests can usually support only a specific functional version of a CRM form. After changing the flow of the tested functionality, a recapture of the test is usually be required.

Creating coded UI tests usually requires more effort for developers but can be more general, making it possible to support even larger structural changes on the tested forms. Coded UI tests can be created either by converting a captured Web Test, or by reusing the existing samples and common components provided by the CRM 2011 Performance Toolkit.

Using automated tests and Visual Studio for tracking requirements and connecting work-items to check-ins provides the ability for bugs and resolution check-ins to be traced back to the specific requirement and test scenario. In fact, with the Visual Studio test framework, it is possible to track code changes and their impacts through to the application and test suites, including which test cases are required to be re-run.

Note: To have this experience across all aspects of an enterprise Dynamics CRM delivery (including CRM customizations) requires an offline build process using the unpacked customization source tree.

Dynamics CRM 2011 Performance Toolkit

The Performance Toolkit for Dynamics CRM 2011 was created by the Dynamics CRM performance team to formalize testing of Dynamics CRM 2011. The Performance Toolkit can be used by the Dynamics CRM partners and customers to collect data to support their CRM deployment decisions.

The toolkit facilitates load testing the performance of Dynamics CRM 2011 deployments. By carefully planning the required dataset that the deployment needs to support and the workload requirements, the toolkit can be used to test if the scale and performance requirements of a particular deployment can be met. This methodology can be used to help with decisions on a particular deployment solution and to avoid costly downtimes at a later stage. The Performance Toolkit for Dynamics CRM 2011 contains the tools (listed below) that can be used in customizing the CRM installation, populating the necessary semantic data for the deployment that is preferred and conducting the benchmarking tests against the CRM installation.

The tools provided in the Performance Toolkit are:

- ImportCustomization Tool
- UserConfigGen Tool
- OrgStructureGenerator Tool
- GoalStructureGenerator Tool
- DbPopulator Tool
- CRM_Perf_Benchmark Tool

The Performance Toolkit for Dynamics CRM 2011 is licensed under the Microsoft Software License Terms. To read the license terms, go to the documentation in the software. The Performance Toolkit contains source code format but does not grant any redistribution rights at all. Therefore, you may use, modify, and add to the source code identified in the CRM_Perf_Toolkit directory for your internal use only.

Knowledge of Microsoft Visual Studio 2010 Ultimate and Microsoft Visual C# is required to use the toolkit effectively.

There are essentially three ways to extend the usefulness of the reference datasets and loadtest the toolkit provides:

- Customize the reference dataset. You can use dbpopulator with custom input files to more accurately reflect a particular customer scenario.
- Modify the reference loadtest to test different user populations, rates of work, and workload composition.
- Create new web tests to cover functionality not included with the reference web tests.

Note: For more information, on the Dynamics CRM Marketplace, see Performance Toolkit for Microsoft Dynamics CRM 2011.

Build Verification Testing

Build Verification Testing involves taking output from a build, deploying to an environment, verifying the deployment succeeded, execution and verification of smoke tests, execution and verification of automated tests for relevant scenarios. Build verification testing usually comprises the following elements:

- **Unit Tests**: Because unit tests are typically the first tests developed, if you are truly using a test-driven development approach you ideally should create them before the code is actually written. By adding unit tests into BVTs during the early stages of a project, you provide at least some code coverage immediately. As the number of functional tests and, hence, test coverage grows, you may opt to remove unit tests from BVTs.
- **Smoke Tests**: End-to-end functional tests that test the basic functionality of your solution. If these fail, something is seriously wrong. These can usually be run relatively quickly.
- **Functional Tests**: These also target end-to-end scenarios, but in this case they test all the scenarios in the project. For large projects it may make sense to categorize your functional tests further (for instance, to enable a particular component to be tested quickly, reliably, and in isolation). These functional tests should be locked down after you have signed off on your solution as being functionally correct.
- **Regression Verification Tests**: Every time a solution bug is found and fixed, regression verification test cases should be added to verify that the bug is fixed and that it is not reintroduced later in the project lifecycle. These tests will typically be cases that were not covered in the functional test cases. You should expect that your regression verification tests will increase in number even after the solution has gone live, if new bugs are discovered and fixed.

On very large projects, you may need to make your BVTs a subset of the full functional test suite (due to length of time they take to execute). For smaller projects, BVTs will constitute the entire set. Obviously, if you are going to integrate the BVTs as part of your daily build, the whole process needs to be automated.

Important: From a BVT execution perspective, a longer deployment step is usually needed for the customization package. So it is more suitable to separate the BVT parts that are able to be executed without deployment from the ones that require it. This way, early build break detection is possible without starting the entire deployment process.

Note: For general information regarding BVT test automation, on MSDN, see How to: Configure and Run Scheduled Tests After Building Your Application.

Environment Automation

Visual Studio Lab Management is an extension of Microsoft Test Manager that helps you to optimize the use of Microsoft Hyper-V technology to manage and use virtual machines in testing, building, and developing applications in Visual Studio 2010. Visual Studio Lab Management is integrated with System Center Virtual Machine Manager (SCVMM) to enable you to manage multiple physical computers that host virtual machines and to manage the storage of virtual machines, virtual machine templates, and other configuration files in SCVMM library servers.

Virtual environments are groups of virtual machines that are managed by Lab Management. Virtual environments enable you to:

- Reproduce the exact conditions of a bug or other development issue.
- Build, deploy, and test applications automatically in a clean environment.
- Reduce the time required to create and configure machines for testing an application.
- Run multiple copies of a test or development at the same time.
- Enable members of a team to create and manage virtual environments without requiring system administrator privileges.

You create an environment by using Microsoft Test Manager and assigning virtual machines to each role that is required for the application that you intend to develop, test, or run. For example, you might be developing a multi-tiered application that requires three roles: a desktop client, a web server, and a database server. By using Lab Management, you can create a virtual environment that assigns a virtual machine to each role, deploys each part of the application to the relevant virtual machine by using Team Foundation Build, and then runs the three virtual machines as a single instance of the application for testing.

Deployment

Manual Development "Deployment"

While not strictly deployment, within the development environment it is possible to push components into the unmanaged layer discretely (not through a solution), which can be broken down to the following:

- Solutions
  - Customizations
  - Dashboards
  - Site maps
  - Ribbons
  - Option Sets
- Plug-in and workflow assemblies
- Web resources
- Custom reports
- Workflows
- Plug-in steps, SDK message processing Steps
- Custom security roles
- Custom base data import
- Other solution elements

There are a number of approaches available to support this approach:

- Metadata service
- Plug-in registration tool
- Developer Toolkit for Dynamics CRM
- CRM UI manual operations

Is should be noted that these actions correspond to the development of CRM components within the development environment and the ability to expose these components within a CRM development organization rather than an approach that should be used for deployment.

Automatic Deployment

For all environments downstream of development, the smallest unit of deployment is a CRM solution package. However, there may be other external components or applications that need to be deployed alongside the CRM solution for the CRM application to function correctly as a whole. It is recommended to automate the deployment process for all downstream environments and use the same approach throughout to test the process that will ultimately be used for production.

A number of automated deployment approaches exist. The most typical for an enterprise production data center are Windows PowerShell scripted installations although some smaller deployments may also still use the Windows Installer, MSI-based installation approach.

Note: Windows PowerShell scripted deployments are the recommended approach to enterprise scale Dynamics CRM application deployments.

Windows PowerShell

Windows PowerShell is a task-based command-line shell and scripting language designed especially for system administration. Built on the .NET Framework, Windows PowerShell helps IT professionals and power users control and automate the administration of the Windows operating system and applications that run on Windows.

Built-in Windows PowerShell commands, called cmdlets, let you manage the computers in your enterprise from the command line. Windows PowerShell providers let you access data stores, such as the registry and certificate store, as easily as you access the file system. In addition, Windows PowerShell has a rich expression parser and a fully developed scripting language.

Windows PowerShell includes the following features:

- Cmdlets for performing common system administration tasks, such as managing the registry, services, processes, and event logs, and using Windows Management Instrumentation (WMI).
- A task-based scripting language and support for existing scripts and command-line tools.
- Consistent design. Because cmdlets and system data stores use common syntax and naming conventions, data can be shared easily and the output from one cmdlet can be used as the input to another cmdlet without reformatting or manipulation.
- Simplified, command-based navigation of the operating system, which lets users navigate the registry and other data stores by using the same techniques that they use to navigate the file system.
- Powerful object manipulation capabilities. Objects can be directly manipulated or sent to other tools or databases.
- Extensible interface. Independent software vendors and enterprise developers can build custom tools and utilities to administer their software.

In a production data center environment, Windows PowerShell can be used with .Net applications developed by the project team to manage the full deployment of the Dynamics CRM application, including for example:

- Import of the Dynamics CRM solutions.
- Import of reference data.
- Registration of plug-ins.
- Deployment of bespoke, custom web services within IIS.

WiX: Windows Installer XML

Windows Installer XML (WiX) is a toolset that builds Windows installation packages from XML source code. The toolset provides both a command line environment that developers may either integrate into their old-style makefile build processes or MSBuild processes from inside integrated development environments like Microsoft's Visual Studio or SharpDevelop to build their MSI and MSM setup packages. The WiX toolset is tightly coupled with the Windows Installer technology. To fully utilize the features in WiX, you must be familiar with the Windows Installer concepts.

The Visual Studio WiX toolset allows you to easily create WiX projects, edit WiX files using IntelliSense, and compile/link your project within the Visual Studio IDE. The WiX Visual Studio plug-in supports VS2005, VS2008, and VS2010. You can create and build Windows Installer packages using WiX within the Visual Studio IDE. Alternatively, you may also use WiX on the command line by calling the tools directly or using MSBuild.

WIX can support your deployment to be able to:

- Package your software into a single-file, double-click MSI for easy installation.
- Read and write to the Windows Registry and create, start, and stop Windows Services during installation.
- Write .NET code that performs specific tasks during installation via custom actions.

Note: For additional information, see http://wix.codeplex.com/ and http://wix.tramontana.co.hu/.

Similarly to the Windows PowerShell approach, WiX is only required as a deployment mechanism when it is necessary to deploy more than just a CRM solution. For example, in enterprise scenarios it is typically necessary to also import reference data, enable plug-ins and activate workflows, and deploy custom web solutions.

Summary

Enterprise-level application delivery is complex and needs strong processes to succeed, and this is continually demonstrated for Dynamics CRM applications.

Through strong processes and a clear understanding of approaches and tooling, delivery of such applications becomes consistent and predictable, leading to reliability, confidence, and a higher quality level within the delivered application.

Appendix A: Acronyms

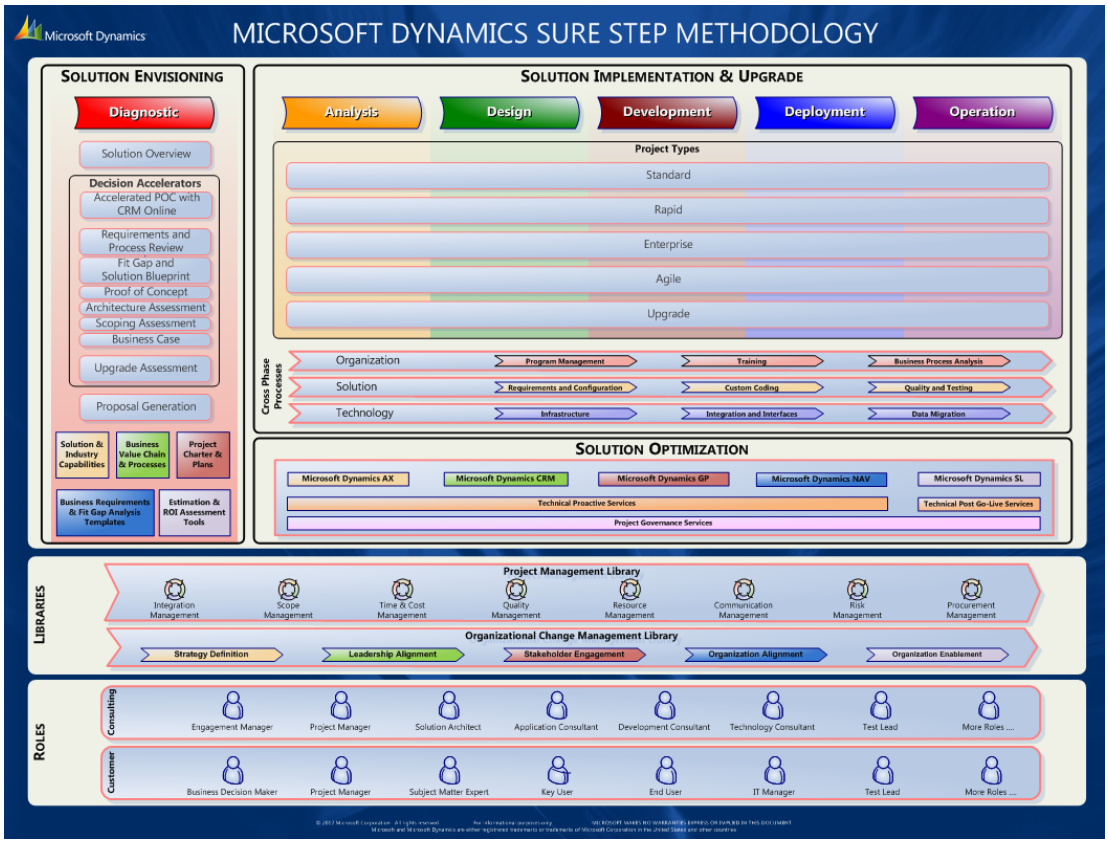
Acronyms related to application lifecycle management for Dynamics CRM solutions are detailed in the following table.

Acronym	Terminology
ALM	Application Lifecycle Management
BVT	Build Verification Test
CI	Continuous Integration
CIT	Component Integration Test
CRM	Customer Relationship Management
IDE	Integrated Development Environment
ISD	Iterative Solution Development
ISV	Independent Software Vendor
MSF	Microsoft Solutions Framework
OOB	Out-of-the-Box
POC	Proof of Concept
RDP	Rapid Deployment Program
SDLC	Software Delivery Lifecycle
TAP	Technology Adoption Program
TFB	Team Foundation Build
TFS	Team Foundation Server
UAT	User Acceptance Test
UR	Update Rollup
WBS	Work Breakdown Structure
xRM	Extensible Relationship Management

Appendix B: Methodologies

Dynamics Sure Step Methodology

Microsoft Dynamics Sure Step is a full customer lifecycle methodology for all Microsoft Dynamics® solutions, providing the Microsoft ecosystem with comprehensive implementations through delivery guidance, project management discipline alignment, and field-driven best practices. Sure Step is designed to enable the solution provider to better serve their customers by helping reduce their Microsoft Dynamics total cost of ownership. Sure Step content covers the Microsoft Dynamics ERP and CRM suite of solutions, including Microsoft Dynamics AX, Dynamics CRM, Microsoft Dynamics GP, Microsoft Dynamics NAV, and Microsoft Dynamics SL. The guidance, tools, templates, and best practices provided in the methodology can help increase the consistency, timeframes, quality, and success of Microsoft Dynamics engagements.



Sure Step is considered a full lifecycle methodology because it encompasses all phases of a customer engagement. Sure Step begins with a Solution Envisioning phase to help customers determine the right solution for their needs. The Solution Envisioning phase is followed by Solution Delivery phases to implement their solution and to provide guidance for the operation and maintenance of the solution in production. For existing Microsoft Dynamics customers seeking to progress their solutions to the latest product releases, Sure Step also provides Upgrade Assessments in the Solution Envisioning phase, followed by Solution Delivery phases to upgrade their solution and then to maintain the production solution in operation.

Sure Step has six phases: **Diagnostic, Analysis, Design, Development, Deployment,** and **Operation**. The Diagnostic phase encompasses Solution Envisioning and provides guidance on product capabilities, including content on focus industries for a corresponding product. The Decision Accelerator Offering is an important part of the Diagnostic phase, designed to reduce the risks and concerns for the customers in their decision-making process for new/upgrade ERP/CRM solutions.

The Sure Step Methodology offers the project types described in the following table:

Project type	Description
Standard	A lean approach for implementing Microsoft Dynamics solutions at a single site.
Rapid	An accelerated approach for implementing Microsoft Dynamics solutions with minimal or no customizations.
Enterprise	A standardized approach for implementing Microsoft Dynamics solutions in complex single-site deployments or in global/multi-site organizations wherein country/site-specific unique business needs have to be factored on top of a core solution.
Agile	An iterative approach to implementing Microsoft Dynamics solutions at a single site requiring specific features and moderate-to-complex customizations. While the Standard, Rapid, Enterprise, and Upgrade project types are waterfall-based, the Agile project type uses the Sprint cycle approach to solution deployment.
Upgrade	An approach to upgrade an existing Microsoft Dynamics solution to a subsequent release of that solution. This begins with a Technical Upgrade to address moving existing functionality to the subsequent release. Any new functionality that is desired can be deployed by using the one of the other project types: Rapid, Standard, Agile, or Enterprise.

Sure Step also features **Cross Phase Processes** that span the project types. A cross-phase process is a group of related activities that span multiple implementation phases in a specific project scenario. The Sure Step Methodology also provides **Optimization Offerings** that feature proactive and post go-live services that are designed to assist the customer and solution provider with an additional level of due diligence in the solution delivery lifecycle.

Additionally, Sure Step provides Project Management and Organizational Change Management libraries, with content to support these key functions in a solution delivery engagement. Sure Step also includes an overview of roles typically involved in an engagement, both from consulting (solution provider) and customer perspectives.

Note: Dynamics Sure Step methodology has a strong delivery guidance and toolset for managing an entire Dynamics CRM project, positioning Dynamics CRM as the main element of the solution and the methodology. Enterprise solutions usually consist of multiple products and even diverse technologies, making it challenging to apply the entire process. Applying the templates and recommendations of Sure Step should be always considered and made part of the specific chosen ALM method to lower the risks and to make the CRM delivery process more transparent. Dynamics Sure Step lacks the guidance regarding the tooling and automation techniques for the specific processes; the tooling should be always selected according to the specific delivery environment and requirements of the solution.

MSF-based Solution Delivery

The Microsoft Solutions Framework (MSF) provides an adaptable framework for successfully delivering information technology solutions faster and with fewer people and less risk while enabling higher-quality results. MSF helps teams directly address the most common causes of technology project failure to improve success rates, solution quality, and business impact. Created to deal with the dynamic nature of technology projects and environments, MSF fosters the ability to adapt to continual change within the course of a project.

MSF is called a framework instead of a methodology for specific reasons. As opposed to a prescriptive methodology, MSF provides a flexible and scalable framework that can be adapted to meet the needs of any project (regardless of size or complexity) to plan, build, and deploy business-driven technology solutions. The MSF philosophy holds that there is no single structure or process that optimally applies to the requirements and environments for all projects. It recognizes that, nonetheless, the need for guidance exists. As a framework, MSF provides this guidance without imposing so much prescriptive detail that its use is limited to a narrow range of project scenarios.

MSF components can be applied individually or collectively to improve success rates for projects such as:

- Software development projects, including mobile, web and e-commerce applications, web services, mainframe, and n-tier
- Infrastructure deployment projects, including desktop deployments, operating system upgrades, enterprise messaging deployments, and configuration and operations management systems deployments
- Packaged application integration projects, including personal productivity suites, enterprise resource planning (ERP), and enterprise project management solutions
- Any complex combination of the above

MSF guidance for these different project types focuses on managing the "people and process" as well as the technology elements that most projects encounter. Because the needs and practices of technology teams are constantly evolving, the materials gathered into MSF are continually changing and expanding to keep pace.

As a framework, MSF contains multiple components that can be used individually or adopted as an integrated whole. Collectively, they create a solid yet flexible approach to the successful execution of technology projects. These components are described in the following table.

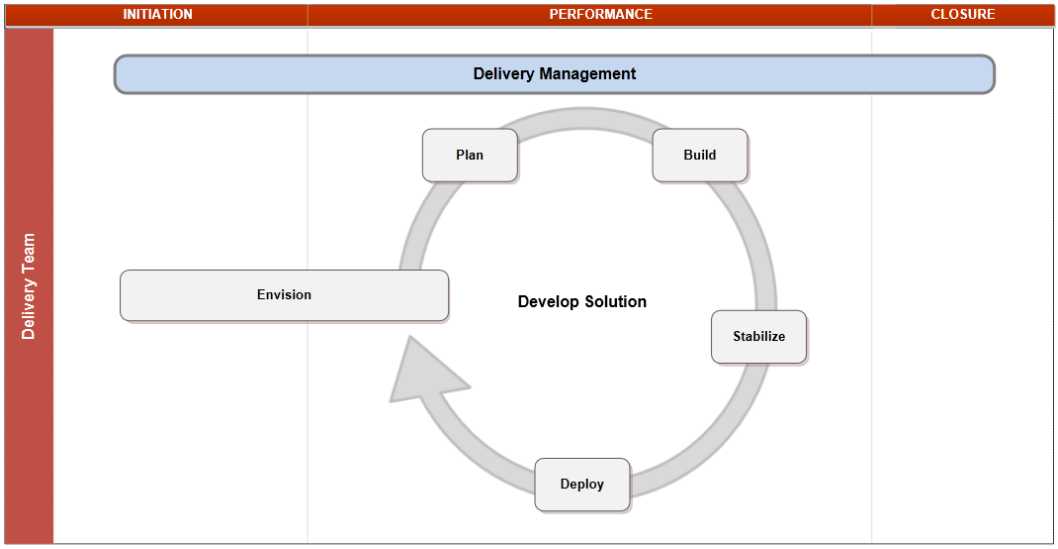
MSF component	Description
MSF foundational principles	The core principles upon which the framework is based. They express values and standards that are common to all elements of the framework.
MSF models	Schematic descriptions or "mental maps" of the organization of project teams and processes (Team Model and Process Model—two of the major defining components of the framework).
MSF disciplines	Areas of practice using a specific set of methods, terms, and approaches (Project Management, Risk Management, and Readiness Management: the other major defining components of the framework).
MSF key concepts	Ideas that support MSF principles and disciplines and are displayed through specific proven practices.
MSF proven practices	Practices that have been proven effective in technology projects under a variety of real-world conditions.
MSF recommendations	Optional but suggested practices and guidelines in the application of the models and discipline.

The MSF Process Model combines concepts from the traditional waterfall and spiral models to capitalize on the strengths of each model. The Process Model combines the benefits of milestone-based planning from the waterfall model with the incrementally iterating project deliverables from the spiral model.

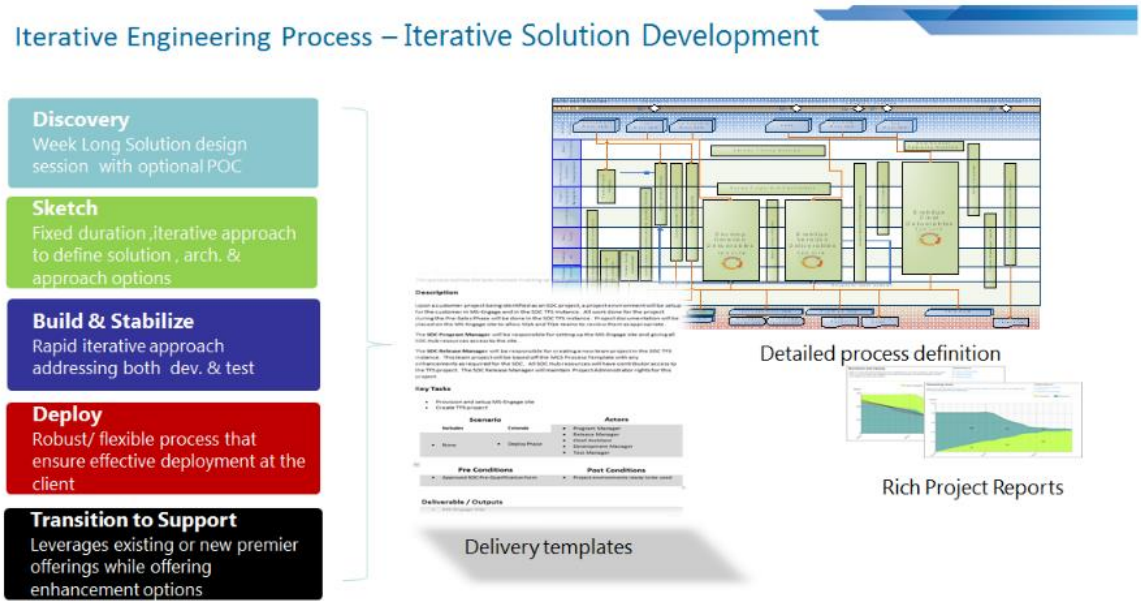
The Process Model phases and activities appear in the following list:

- **Envision**: Describe the solution concept and define the project team necessary to deliver it.
- **Plan**: Assemble detailed plans and designs necessary to deliver the solution.
- **Build**: Construct a solution that includes all aspects of the project needs.
- **Stabilize**: Polish and verify that the solution meets customer and user needs and expectations.
- **Deploy**: Deploy and integrate the solution to its intended production environments.

The MSF Process Model is depicted in the following graphic:



Iterative Solution Development



Iterative Solution Development (ISD) is a methodology used to reduce solution delivery risk and highlight Microsoft's deep experience building custom application-development solutions. ISD enables on-going productive customer feedback, a single system of record for improved traceability, and consistent guidance on tools and application development recommended practices.

- ISD is recommended for extremely complex, custom-development solutions.
- ISD is the Microsoft Services Solution Delivery (SSD) approach for envisioning, planning, stabilizing, and deploying complex custom application development solutions.
- ISD is derived from Services Delivery Methodology (SDM), which is based on Microsoft Solution Framework (MSF) and sourced from the World Wide Solution Development Centers. ISD is used when Microsoft is the prime contractor for large, complex, and custom application development engagements.
- ISD leverages five core pillars to ensure delivery: a team model, a mentoring model, a process model, a governance model, and guidance focused on management of the development environment (built on TFS).

ISD Phases

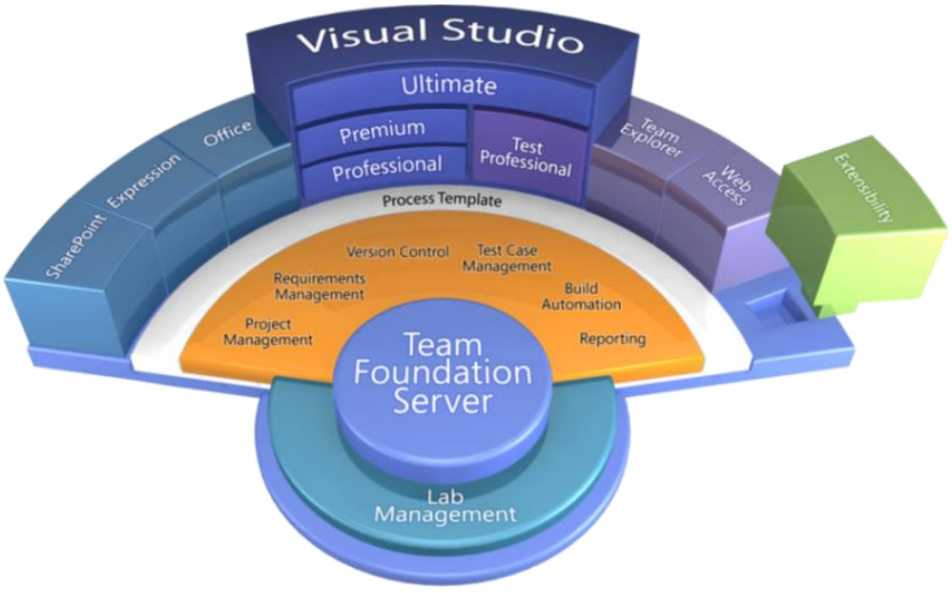
- The ISD **Discovery Phase** provides detailed guidance on all technical and business pre-sales activities required to win large, complex, and custom application development Tier 1 deals.
- The ISD **Sketch Phase** provides detailed guidance on all of the activities required to successfully deliver a solid statement of work (SOW) for the Build, Stabilize, and Deploy phases. The ISD Sketch Phase is designed to be used on large, complex, and custom application development Tier 1 engagements.
- The ISD **Build & Stabilize Phase** is the process of constructing the solution for the customer. This is the main Delivery Management phase. The iteration plan developed during Sketch is executed delivering working features and capabilities for each Release. It is built on the ISD team model and leverages TFS tooling and automation, covering development and test processes, source and version control policies, and testing and QA methods to ensure the ISD goal of high quality solution.
- The ISD **Deploy Phase** provides detailed guidance on all the activities required to successfully release custom application development solutions into production. The ISD Release Phase is designed to be used on large, complex, and custom application development Tier 1 engagements.
- The ISD **Support Phase** provides detailed guidance on all the activities required to successfully support custom application development solutions once they have been deployed into production. The ISD Support Phase is designed to be used on large, complex, and custom application development Tier 1 engagements.

TFS Usage

The ISD process template is a modified version of the TFS 2010 MSF for Agile 5.0 process template. Changes to support ISD have been purposely limited so that all out-of-box reporting and documentation for the default Agile template still applies.

Visual Studio Team System: Application Lifecycle Management

Microsoft Visual Studio Team Foundation Server is the collaboration platform at the core of the Visual Studio solution for application lifecycle management. Team Foundation Server provides fundamental services such as version control, work item and bug tracking, build automation, and a data warehouse. Powerful reporting tools and dashboards provide historical trending and visibility into overall project health, and real-time metrics give early warnings of potential problems so that you can make data-driven decisions and course corrections. In addition, agile planning tools and integration with Microsoft Project and Project Server help you plan and manage your projects.



You can apply proven practices to manage your application's lifecycle by using the suite of tools in Visual Studio Premium and Visual Studio Ultimate in combination with Visual Studio Team Foundation Server. By using these tools, your team can better understand customer needs and more effectively design, implement, and deploy code. For example, your team can trace requirements to checked-in code, builds, and test results. By adopting these practices, your team can create software that your customers value more and that is faster and more reliable. You can use these tools to achieve the following results:

- Plan and track your project. Enact processes and monitor their quality to help your team turn customer requirements into working software.
- Design functionality, either on top of existing assets or from scratch, by using architectural diagrams to communicate critical information about your team's software.
- Write, unit test, debug, analyze, and profile your application by using tools that are integrated with the rest of the application lifecycle so that your team can understand how your progress contributes to the project. Use version control to manage your source code and other files.
- Build your application by using an integrated build system so that your team can ensure that quality gates have been met and verify that requirements have been fulfilled in each build.
- Test your application by running manual or automated tests, including performance and stress tests. Manage testing systematically so that your team knows the software quality on any given day.
- Deploy into virtual environments to enable more sophisticated development and testing.

Visual Studio 2010 Ultimate Features

Microsoft Visual Studio 2010 Ultimate package contains all tools that support the entire application development process from start to finish. Teams can realize increased productivity and cost savings by utilizing advanced collaboration features as well as integrated testing and debugging tools that help ensure that you deliver high-quality code every time. The features cover all aspects of enterprise development:

- ALM
- Debugging and diagnostics
- Testing tools
- Architecture and modeling
- Database development
- Integrated development environment
- Development platform support
- Team Foundation Server
  - Build management
  - Version control
  - Test case management
  - Work item tracking
  - Reports and dashboards
- Lab Management

Note: For more information, on MSDN, see Application Lifecycle Management with Visual Studio and Team Foundation Server or the white paper Introducing Visual Studio 2010.

Application Lifecycle Management Strategies

For detail regarding general ALM strategies and a comparison of ALM and the Software Development Lifecycle (SDL), see the white paper What is Application Lifecycle Management (ALM)?.

Appendix C: Aspects of Enterprise Project Delivery

Dimensions

- Length, iterations
- Affected organization size, user base, Multi-X
- Requirement, fit-gap complexity
- Integration level and type
- Solution architecture complexity
- Existing functionality and data—upgrade/migration
- Reusability—future, existing IP
- Development team and phase complexity—parallel, customer-supplier, IT-business
- Project structure, management, delivery methodology, risk management
- Environmental constraints—technical, geological, human, natural

Challenges

Dimensions	Challenges
Project length, number of iterations	Version management
	Change management
	Release management
Organization size, user base	Multi-site management
	Multi-tenancy
	Multi-language
	Multi-national
	Scalability (project, solution)
Requirement, fit-gap complexity	Requirement management
	Requirement structuring and packaging
	Requirement scheduling
	Requirement—work item mapping and tracking
	Requirement and feature validation and testing
Integration level and type	Integration design
	Integration schema
	Development and versioning
	Integration testing
	Solution component packaging and versioning
Solution architecture complexity	Heterogeneous build environment and management
	Layer separation/decomposition support
	Deployment scenarios, combinations and complexity
Existing functionality and data	Upgrade
	Migration
	Integration

Dimensions	Challenges
Reusability	Existing functional/architectural elements
	Creating common elements, refactoring support
	Parent-child component and package management
	IP generation in reusable format
Development team and phase complexity	Multiple developer support
	Multiple development teams
	Agile work-item management
	Parallel feature/package development
	Feature and solution branch management
	Development/operations/business/management team support
	Development environment and infrastructure management
	Development process support
	Build process support
	Customer team support
	Third party external team/external package support
Project structure, management, delivery methodology, risk management	Project size scalability
	Feature change management
	Resource change management
	Project delivery methodology support
	Monitoring support
	Delivery QA support
	Risk management support
	Decision making and tracking support
Environmental constraints: technical, geological, human, natural	Technical platform diversity
	Multi-site support
	Cloud environment
	Remote teams
	Remote access
	Staging constraints (data, access, etc.)
	Multi-TZ teams
	Multi-language teams
	External dependencies

Capabilities

Group	Capability
Dynamics CRM 2011	Solution management
	Solution versioning
	Solution layering
	Unmanaged and managed solution support
	WYSIWYG GUI for customizing CRM entities, relationships, forms, views, charts, dashboards
	GUI for authoring workflows and dialogs
	Multiple API support (WCF, .Net, SQL) for extending functionality
	Administration UI for security, localization settings
	Management UI for base data
	Framework to import 3rd party components and solutions
Dynamics CRM Developer Toolkit	Visual Studio solution and project template
	Native support for any Source Control plug-in that works within Visual Studio 2010
	Integrated CRM customization UI into VS environment
	VS component build and deployment into a target CRM solution
	Exported CRM Solution as part of the build and deploy process
SolutionPackager	Unpacking the solution xml files to customization elements
	Packing the customization elements to a single solution package
	Supporting managed and unmanaged solutions
	TFS automatic build process integration
	Pluggable framework to add processing steps to specific customization elements
GUI editors	SiteMap editor
	Ribbon editor
TFS	Requirements management
	Build management
	Version control
	Bug tracking
	Test management
	Development reporting
	Team development, team management
	Source control and branching
	Development and code policies
	Localization
	QA process support
	Integrated Development Environment (VS)
LocStudio	Localization management

Appendix D: CRM Solution Concepts

Solutions and layering are fundamental concepts within CRM that need to be fully appreciated and understood to construct an approach to lifecycle management for Dynamics CRM applications. There are three key concepts that need to be introduced:

- Solution Packages: Act as containers for functionality required to be deployed as a unit
- Layers: The consequence of a specific component being affected by change within one or more solutions
- Managed Properties: The mechanism to control how layers interact with each other

Note: This appendix provides a "quick reference" overview of more detailed information that is presented in the Dynamics CRM 2011 SDK, in the section Package and Distribute Extensions.

Solution Packages

Solution packages are the container mechanism to transport Dynamics CRM configuration and customization between organizations. There are two types, unmanaged and managed.

Unmanaged

- Contains only customizations from the unmanaged layer
- Will not include dependent components
- Imports only into the unmanaged layer:
  - Ideal during development
  - Everything within could be changed
  - Overwrites existing customizations
- Contains a complete copy of each component

Managed

- Will get its own layer on import.
- Analogous to an installer ".msi" file; it is a distribution package.
- Contents cannot be directly changed.
  - This does not mean DRM.
- Components may be customized where managed properties allow.
- Contains only deltas for component that supports merging (FormXML, Ribbon & Sitemap).

Layers

Layers should be considered on a per-component basis. Although typically drawn to convey the effect of a solution on another solution, this is always at a component level.

Unmanaged

- There is only one. It always exists.
- All customizations made on this server/organization will reside here.
- "Solutions" exist here as containers.
  - Ownership by solutions is weak; by-reference only.
  - No independence between solutions. If one component exists in two solutions, they both see the current state of the component equally.
- New customizations made will trump all prior customizations. (They override the lower layers.)
- Customizations cannot be undone, but they can be deleted.

Managed

- There can be many:
  - Always one for the “system” solution.
  - Discrete layer for each managed solution package imported (at a per-component level).
- Only created by importing a managed solution.
- Time of import sets order of precedence.
- You cannot customize components inside the layer.*
- They can be deleted, replaced, and versioned.
- Deleting a layer may delete data.

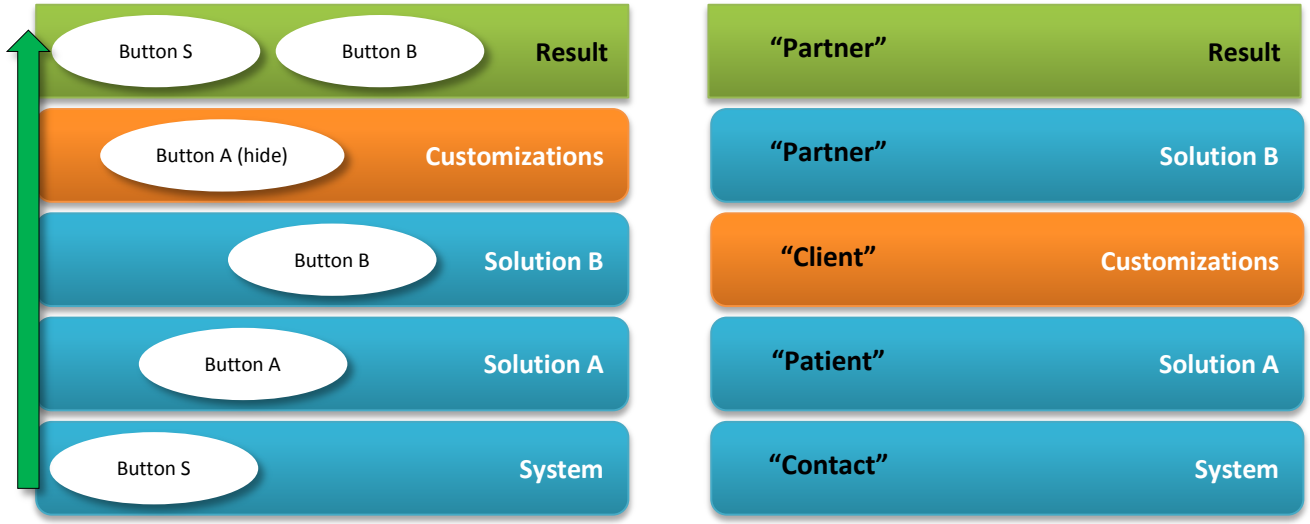
Managed Properties

Managed properties control how layers interact with each other; they control the level of customization achievable on top of components from a managed solution that has been imported. After releasing a managed solution, the managed properties can only be changed to reduce how restrictive they are.

- Control order of precedence (which customizations to allow on top of an imported solution’s managed layer).
- The ability to customize cannot be removed or disabled during solution update.
- The ability to customize can be enabled during solution update.
- The default ability to customize is fully customizable.

In general, if the consumers of the solution are not known or trusted or the changes that may be made could break the solution, it is recommended to lock down the managed properties for the solution. These managed properties can be opened back up to enable specific components to be customized at a later date as needed.

Merge Behavior



Updating Layers

Unmanaged

New customizations trump all prior customizations, overriding the lower layers.

Important: Changes applied by importing an unmanaged solution cannot be uninstalled. Do not install an unmanaged solution if you want to roll back the changes.

Managed (No Overwrite)

- Customizations in unmanaged layer are preserved.
- Importing newer versions creates new managed layers directly above the previous version's managed layer.
- Importing the same version replaces contents within the existing layer.
- Importing cannot remove pre-existing components.
- Generate and import a minimal solution to "hotfix" a larger solution.
- Reports, E-mail templates, and plug-in assemblies skip updates if they are not the "top" layer.

Managed (Overwrite)

The recommended approach is to always preserve customizations (no overwrite). If the updates are mandatory for the solution to function appropriately, then overwrite is needed:

- Replaces the content of the "unmanaged" layer and makes the managed solution the top layer.
- Ensures that updates included in the solution are effective.
- Overwrites all customizations and should be used with caution.
- Greater use of managed properties makes this approach less necessary.

Dependency Tracking

The solutions framework automatically tracks dependencies across components.

- The following operations perform dependency checks/operations:
  - Individual components: CRUD, Add Existing (to a solution).
  - Solution: Import, Export, Delete (uninstall).
- Dependencies are version agnostic. As long as the unique name/id of the component and the package type matches, a dependency is considered valid.
  - Managed components cannot depend on unmanaged components.

Shared Publishers

- Components in managed layers will be owned by the solution publisher.
- Publisher owns the component, not the solution.
- Components with same name and publisher will be considered the same thing.
- Removing a solution does not remove a component when it is referenced by another solution using the same, shared publisher.
- Be wary of predictable names and collisions.
  - For web-resources, create names that imply virtual directories

Appendix E: Managing Complex CRM Scenarios by Using the SolutionPackager Tool

In enterprise scenarios, it is a common requirement to be able to deliver common or separate CRM modules to separate business units according to an enterprise-level or separate roll-out timeline.

The typical challenges of this scenario result from parallel development tracks and frequent changes to enterprise projects, which require the greatest degree of flexibility in every aspect:

- Sharing common code/customization base
- Managing solution layering and dependencies
- Managing localization
- Managing changes in layering (moving items to common or back, customizing common elements locally)
- Versioning common and local solutions
- Parallel development on multiple solutions, multiple solution-layer setups, and multiple versions of a solution
- Central repository of every/any solution
- Test and deploy all possible or required combination of solution layers
- Managing solution branch and merge
- Regression testing changes and impacts across solutions and branches
- Common, single versioning of all solution elements

Development and Version Control

You need streamlined and common techniques of development and repository management for complex enterprise deliveries. The standard features of Microsoft Team Foundation Server serve as the basis of larger application delivery management scenarios:

- ALM
- Debugging and diagnostics
- Testing tools
- Architecture and modeling
- Database development
- Integrated development environment
- Build management and automation
- Version control
- Test case management
- Work item tracking
- Reports and dashboards
- Lab management

When working with multiple development teams developing multiple solutions and versions in parallel on an enterprise delivery, the following base process is minimally recommended:

- Developers should have isolated VMs to work on either hosted locally or centrally to share infrastructure between team members. Every developer work-item needs to be committed to the TFS environment (integration environment).
- The development lead is in charge of making sure that the developments (customizations, configurations, and custom source files) can integrate in the integration environment without any side effects.
- The TFS source control should be considered as the central repository for **all** solution items:
- All CRM customizations (Entities, forms, views, workflows, templates, web resources, etc.)
- Base data
- Source codes and projects for custom components
    - Plug-ins, custom activities
    - Custom web applications
    - Custom .Net components
    - Custom services and other external components
    - Web resource files

- External components and references (SDK, Enterprise Library, etc.)
- Unit test source codes for above components
- Automated test sources for the entire application
- Setup package for the entire application
- Deployment scripts for the packages
- Build definition for the entire application

The customer requirements and the work-item break-down should be stored in TFS work-item management. This way the development check-ins can be connected to the work-items, and the built-in development tracking and reporting functions of TFS can be leveraged. You will also be able to use the change-impact analysis functions of TFS for risk analysis and selecting the regression test scenarios for a specific change-set.

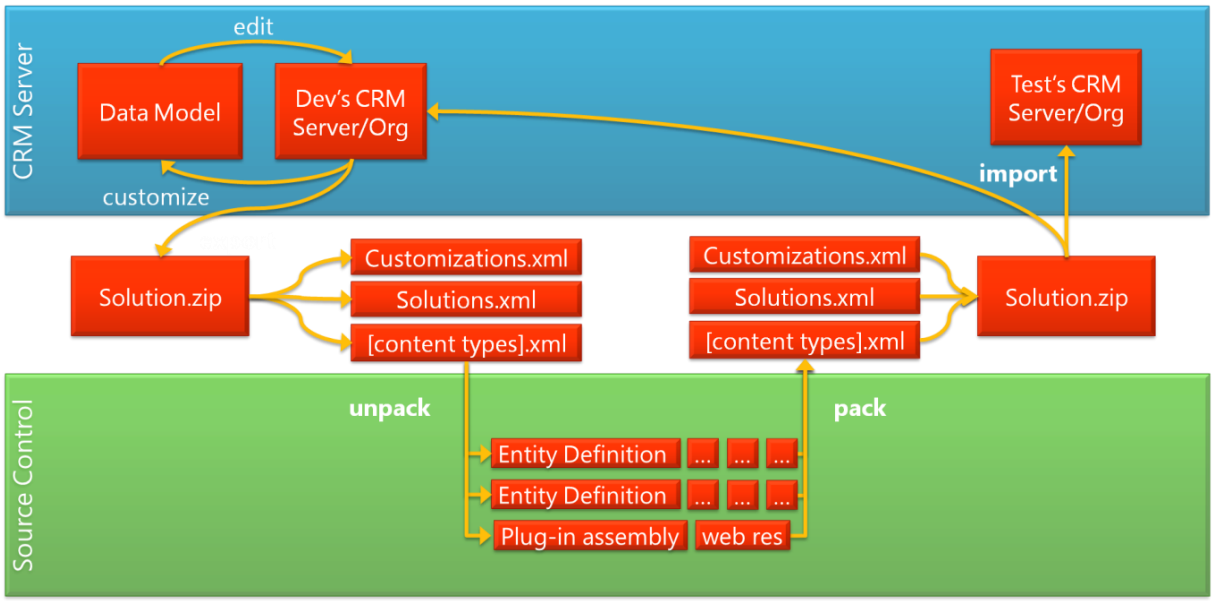
In multi-feature and multi-team scenarios you usually need multiple development and test environments, probably using different versions of the application. The TFS build automation and lab management functions provide the necessary toolset for managing daily build and deployment steps. You find more details about the topic in the Build and Test sections.

Version Control Process

The CRM customizations are out-of-the-box bound to a single CRM solution zip file, which requires special techniques to manage as part of a source control system.

The SolutionPackager tool (see Use the SolutionPackager Tool to Compress and Extract a Solution File) is designed to unpack (split) a solution .zip file into a broad set of files within folders by entity name. The breakout into files enables that modifications to different components will not cause changes to occur in the same files. For example, modifying an entity Form and a View would result in changes in different files (hence no conflicts to resolve). Pack will take the files from disk and glue them back together to construct a managed or unmanaged .zip file.

The development process for Solution Packages with the SolutionPackager tool is illustrated in the following graphic.

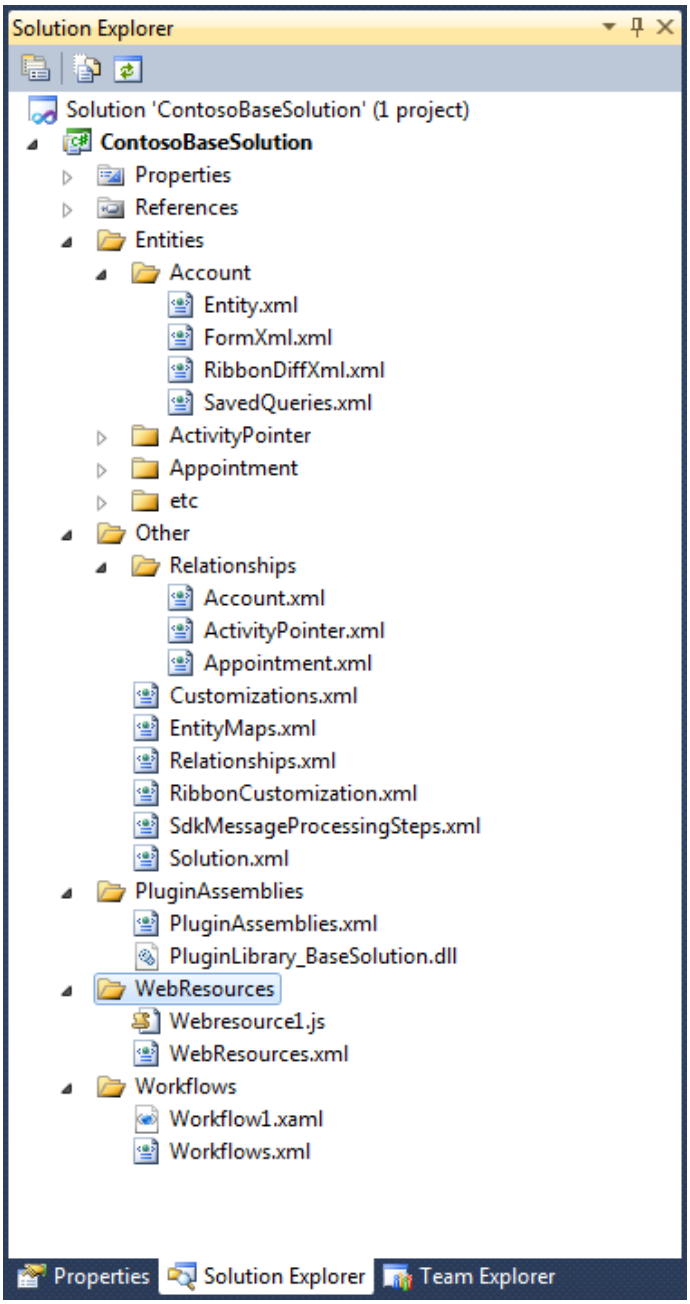


The developer works on his/her daily environment to work on the CRM development task allocated. Before starting a work item the developer (or the build master) gets the latest version of customization tree from TFS and uses the SolutionPackager tool to create the solution zip file. The developer (or the build master) deploys the latest solution package on the development environment. The same solution can also be deployed to the test CRM environment.

At the end of a day or after finishing a work item, the developer exports the solution zip file from the development environment. Executing the SolutionPackager tool on the zip file will generate the separate customization elements of the entire solution. The customization files need to be checked out and checked-in individually only when changed.

Solution tree structure

The unpacked solution structure is illustrated in the figure below.



One Visual Studio project per solution

Entities

- One folder per entity
- Entity schema split by:
  - Entity definition, attributes
  - Form definitions
  - Ribbon definitions
  - Views, SavedQueries
- Entities tree only contains elements from the specific solution

Other elements

- Relationship list
- Relationship definitions (per entity)
- Customizations (base structure of the full schema to pack/unpack)
- EntityMaps
- RibbonCustomization (global ribbon definitions)
- SdkMessageProcessingSteps (registered steps for assemblies)
- Solution.xml (ImportExportXML with SolutionManifest)

PluginAssemblies

- Assembly registrations
- Assembly binaries

WebResources

- WebResource files and binaries
- WebResource definitions

Workflows

- Workflow definitions
- Workflow CRM registrations (not step registrations)

The Visual Studio project may be created manually. The SolutionPackager tool will automatically create the folder structure below that (the unpack script need to be executed in the solution project folder).

Manual Steps

The current version of SolutionPackager tool provides the basic functionality to convert the packaged customization zip file to a version controllable folder structure. The following manual steps are still needed by the developers or test leads to be able to maintain the source tree:

1. Check-in check-out of specific customization elements, which will be affected by a work-item change.
2. Execute the pack/unpack operation manually and selecting the changed customization elements (Note: this can be integrated into the clean and deploy process of the Developer Toolkit).
3. Taking care of the managed and unmanaged version of components at check-in time.
4. Managing the assembly versioning (FQ strong names) of plug-ins used by a CRM solution.
5. Managing RibbonDiffXML (merging changes).

The following techniques may be used to further support the version control process:

- Removing assembly strong names from customization files to be able to track only the real changes.
- Some schema parts are reordered randomly during solution import-export; these elements may be ordered alphabetically to more convenient source control tracking.

Localization Challenges

Enterprise CRM projects usually face the requirement of supporting multiple locales. Although CRM has OOB support for localization, it is always a challenge to provide full solution localization because different components required different localization strategies, and the explosion of components may be required.

Localization of some specific CRM components is currently not supported or not applicable. For example:

- Workflows
- Dialogs
- Templates
- Roles & FLS profiles
- Plug-in assemblies

CRM Developer Toolkit

The CRM Developer Toolkit may be used for daily developer work for creating streamlined plug-ins, generating codes, solutions, and accessing CRM customization pages. The source code should be checked in directly under TFS. The applied customizations can be exported, unpacked from the CRM organization, and the specific changed elements can be checked in manually.

Note: The Developer Toolkit can support the process of packing and unpacking by turning on the ability to export a CRM Solution as part of the local deployment process.

Sitemap, Ribbon Editor

The CRM customization system only supports editing the sitemap and ribbon definition manually as xml files, after a manual export and followed by an import operation. The SiteMap and Ribbon Editor tools provide a graphical UI for executing these types of customizations.

These tools can be also combined with the unpacked structure:

- Sitemap.xml is unpacked into a separate xml file and can be directly edited by SiteMap Editor.
- The Ribbon Editor supports editing directly on a CRM environment, so developers need to edit on the server and then unpack the solution zip file and check-in the changes.

Daily Developer Work

The developers should use unmanaged solutions and shared configuration environments during development time. One developer usually works on one solution (either CRM or other Visual Studio solution), but multiple developers can work on the same solution in parallel.

The developer environments should be typically updated on a per-day basis using the last successful build. You should consider the best environment allocation and setup strategy for your team, depending on the actual project and team setup (*see Deployment and Test*).

The developers typically use standard CRM customization UI for executing their daily work. They create CRM customization elements (entities, screens, ribbons, views, dialogs, plug-in registrations, etc.). They may use Visual Studio for creating custom developed plug-ins, web applications, or other components, and may also use other external tools such as resource editors, designers, sitemap, ribbon, and metadata editor for CRM.

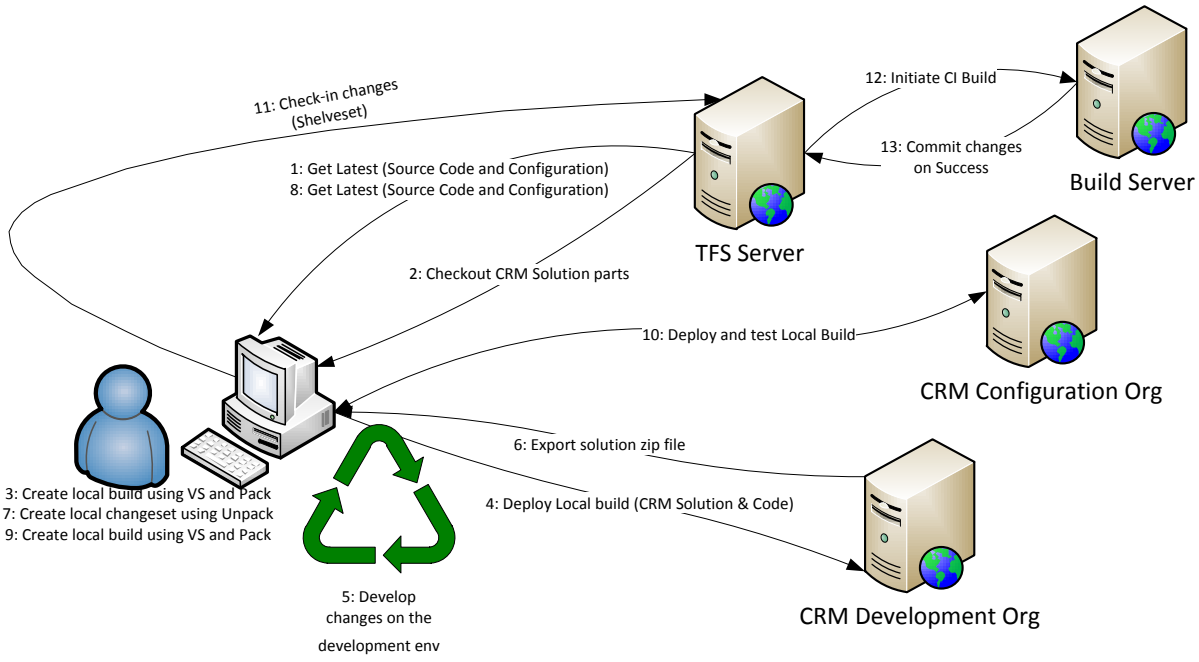
The editing of external components and resources can be typically executed offline on the developer machine after executing the TFS check-out operation (VS built-in editors usually manages check-out automatically).

Just before the check-in, they need to export the solution from the CRM environment and execute a special batch file per solution, which will unpack the customization file.

Developers need to take care of the actual check-in. They need to be aware of the executed changes and which schema members it affected. Only the changed elements need to be checked-in, and there are also special cases when the check-in elements need also to be merged back or compared to the previous version (for example, ribbonDiffXml).

Before the actual check-in operation, the developer must get the latest sources and create a local build, including a solution package, to deploy and test in the development environment; this is the recommended practice for quality check-ins. This process may be also supported by the Gated Check-in or Continuous Integration feature of TFS.

The daily development process is illustrated in the following figure.



Build Automation

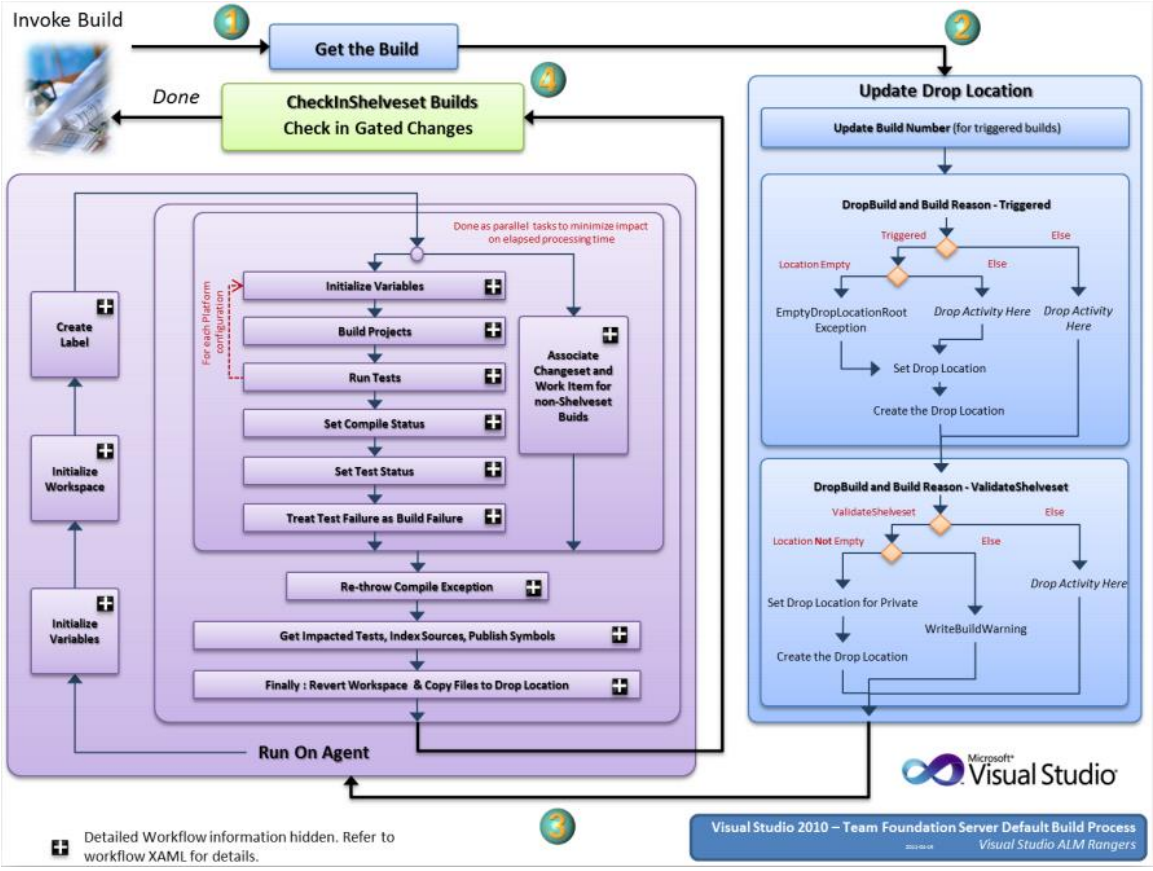
The build process can be automated using the standard TFS build agent. The build agent is a Windows service that executes the processor-intensive and disk-intensive work associated with tasks such as getting files from and checking files into the version control system, provisioning the workspace for the builds, compiling the source code, and running tests.

A typical build process consists of the following steps:

1. Getting the build definition (You may have multiple parallel projects/builds.)
2. Update build number (solution versioning)
3. Prepare build context
    a. Create drop location
    b. Get build agent
    c. Get build directory and initialize workspace
4. Get source files
5. Compile sources and create setup packages
6. Run unit tests
7. Process test results
8. Drop build and notify

The build process may be extended further by automated deployment and testing (for more details see the Build, Test and Deployment sections):

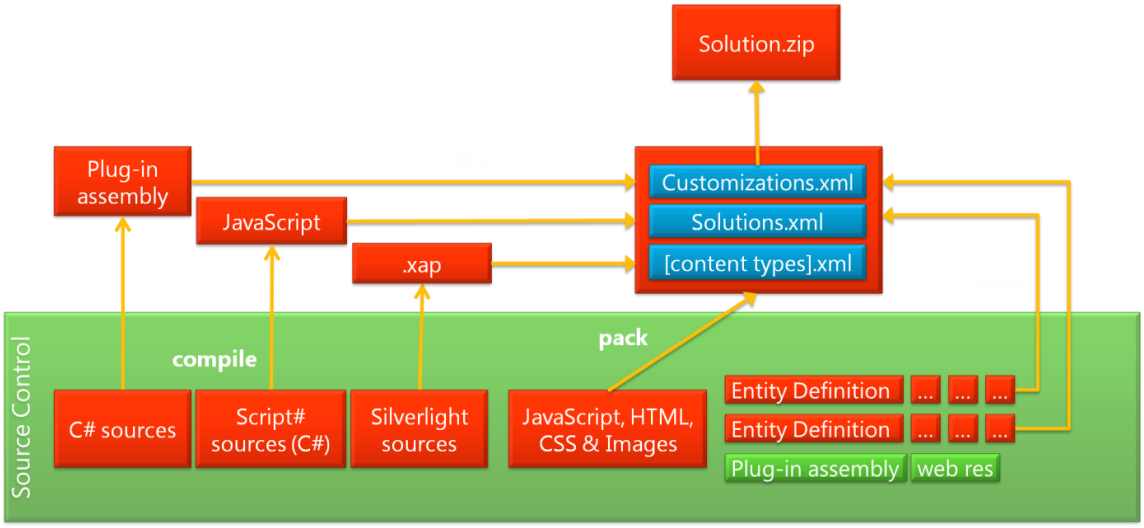
- Rollback test environments
- Deploying setup packages to test environments
- Executing automated test steps/schedule manual test scenarios



Offline Build Process

The offline build process is designed as an independent build flow enabled to execute the building of a whole CRM solution package without a running instance of Dynamics CRM. The offline build process leverages the static schema of the solution elements and customization files (see Development and Version Control).

The build process can be easily extended to use the SolutionPackager tool. The CRM Solution Builder project template and target file samples can be used to integrate the pack operation into the daily build process (see CRM Solution Builder).



During the build process, the Visual Studio standard projects will be compiled, built, and dropped as defined in the build and project definitions.

Building the CRM customization project will trigger the pack operation will the up-to-date customization source tree (see solution tree structure above). All customizations will be merged together as a single customization.xml and solution.xml file. The previously built solution resources and assemblies can be included directly in the solution package.

Note: ILMerge operation may be required to make DB-registered assemblies work with referenced assemblies.

Automating Deployment

The deployment automation of packed CRM solution can be executed identically to standard CRM solution deployment.

On enterprise projects the following scenarios can cover the typical deployment requirements:

- Automated deployment of development environments, test environments, and integration environments
- Offline setup packages and scripts to enable deployment to acceptance and production environment as a sandbox
- Rollback environments to a baseline (both CRM components and the CRM database)
  - May already contain existing solutions and possible CRM Update Rollups
- Deploying CRM solutions
- Deploying CRM base data and custom security roles
- Additional configuration of organization, BU, and user settings using the CRM API

Note: For further details on CRM automated deployment process see the Test and Deployment sections.

Managed/Unmanaged Solutions

The version control differences between managed and unmanaged solutions are already described in Managing Version Requirements section. The SolutionPackager tool will process and unpack both types of solutions. The unpack process will create the differencing elements with “managed” postfix for easier version control management.

The general recommendations for using managed or unmanaged solutions:

- Use unmanaged solutions for development.
- Use managed solutions for all downstream environments.
- Use as few solutions as possible for easier management.
- Customize existing managed solutions as little as possible.
- Execute integration tests using managed solutions frequently to test and address possible collisions.
- Test managed solution combinations for every possible or supported scenario.

Appendix F: Development Environments

General

There are a number of topologies that you can use when setting up a development environment for Dynamics CRM. These topologies range from a single Dynamics CRM deployment with a single organization to a Dynamics CRM deployment hosted on a Virtual Machine instance per-developer.

Single Instance, Single Organization

In this scenario, each developer would work with Visual Studio locally, connecting to a shared instance of Dynamics CRM with a single organization. Since developers will sometimes need to trial concepts with functional configuration of CRM, this environment could be extended to provide two organizations such that all developers would work against the "Development Organization" when developing/debugging their code and trialing new configuration concepts; however, "real" functional configuration would be performed on the "Configuration Organization" to be kept clean and consistent. Using a second organization for configuration also means we have logical consistency across all of our scenarios.

Pros

- Low infrastructure and licensing cost
- Low effort for setup and maintenance of developer environment
- Immediate start for new developer joining the team
- Same basic Dynamics CRM 2011 setup (Dynamics CRM 2011 Update Rollup packages, Dynamics CRM 2011 Update Rollup packages language packs etc.) and same data for all developers (basic test and sample data)

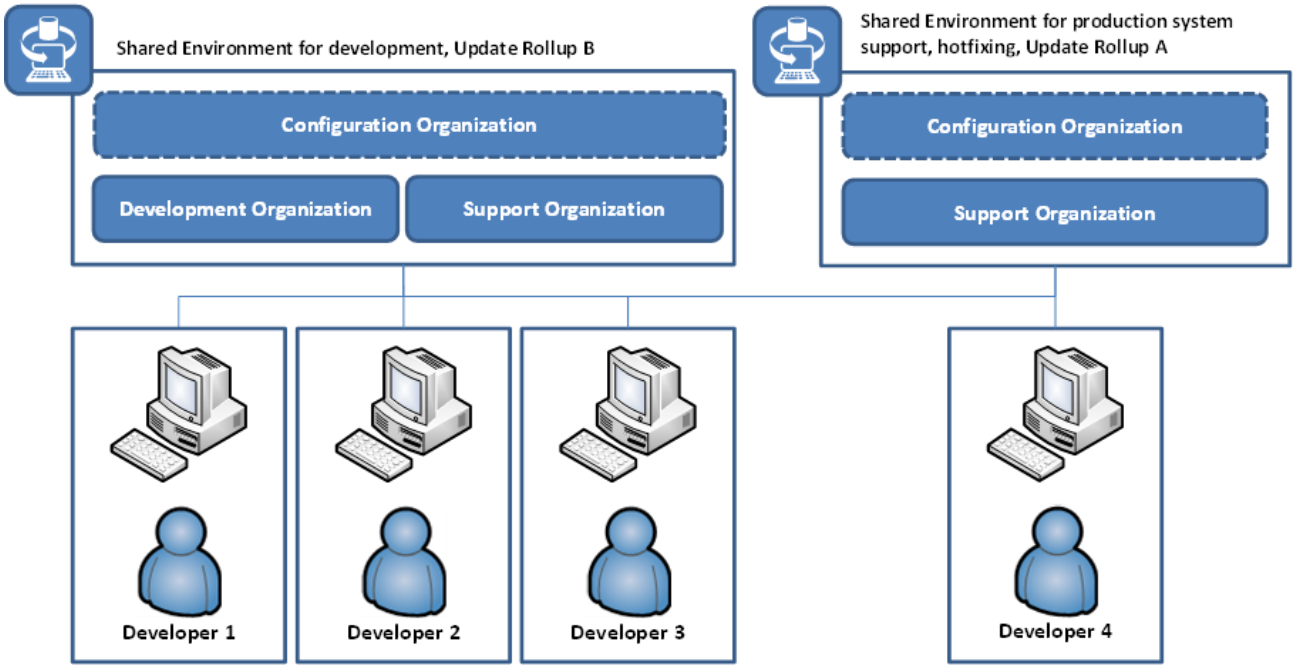
Cons

- Lack of developer isolation.
  - Need to attach to same processes in case of debugging.
  - "Noise" from web service calls made by other developers.
  - A breaking defect breaks all developers.
  - Capturing and reverting to a known state affects all developers and has to be a decision in collaboration with the co-developers.
  - All developers need to be working on the same project phase. Conducting work on the v-next project phase in parallel with the current project phase will contaminate the solution. Development of production hotfixes will be challenging if different Rollup Packages or other changes in the environment must be considered.

Regarding the scenario at Contoso Diagnostics from the perspective of the Core solution development team:

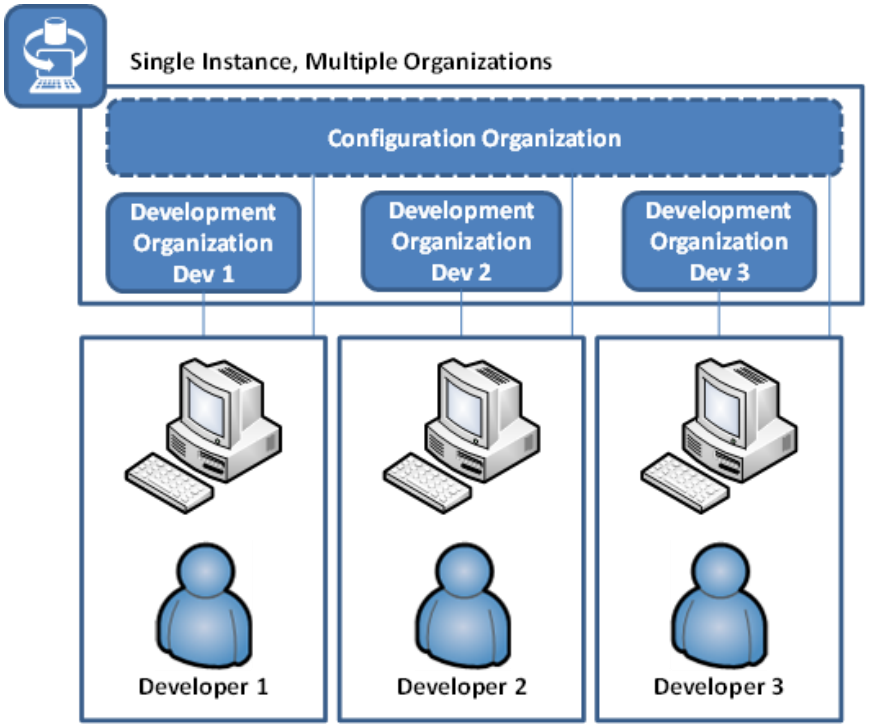
- To support simultaneous development of hotfixes for the current running production system with version 1 of Contoso's core solution and development of a next version of Contoso's ore solution, an additional CRM organization for hotfix development may be required.
- If there are different Update Rollup versions in development and production environments, an additional development environment infrastructure with a separate Dynamics CRM installation may be required.

Both options are shown in the following diagram.



Single Instance, Multiple Organizations

This scenario extends on the former and isolates each developer's work from one another. Each developer has his/her own Dynamics CRM Organization to deploy components to and perform local configuration to prove concepts. All functional configurations that will be retained are conducted on a shared "Configuration Organization" purely used for this purpose.



Pros

- Low infrastructure and licensing cost.
- Partial developer isolation.
  - Configuration and code changes.
  - "Noise" from web service calls made by other developers while debugging.
- Work in the different project phases possible (for example, work on v-next phase and hotfix phase if all depend on same Update Rollup level).
- Ability to (reset) snapshot the database (backup/restore or detach/reattach DB files from source controlled known state); this is useful to clean out trial changes.
- Central configuration organization is kept clean from trial development work.
- Immediate start for new developer joining the team.

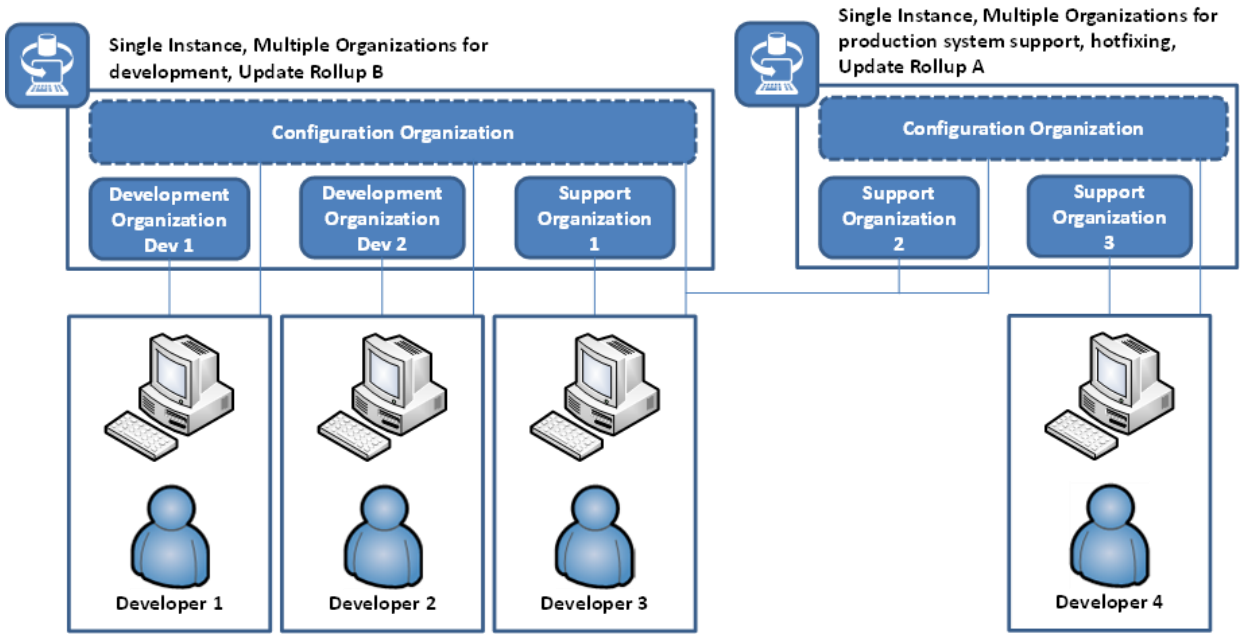
Cons

- "Noise" in CRM platform trace that is shared across organizations.
- All organizations need to be at same Update Rollup level; if there are code changes as well as DB schema changes, this would exclude the "pro" to rollback an individual developer's organization database.
- More complex process; developers must export the CRM solution from their central configuration organization and import it into their personal development organization for current configuration changes.

Regarding the scenario at Contoso Diagnostics from the perspective of the core solution development team:

- To support simultaneous development of hotfixes for the current running production system with version 1 of Contoso's core solution and development of a next version of Contoso's core solution, an additional CRM support organization for hotfix development may be required.
- If there are different Update Rollup versions in development and production environments, an additional development environment infrastructure with a separate Dynamics CRM installation may be necessary.

Both options are shown in the following diagram.

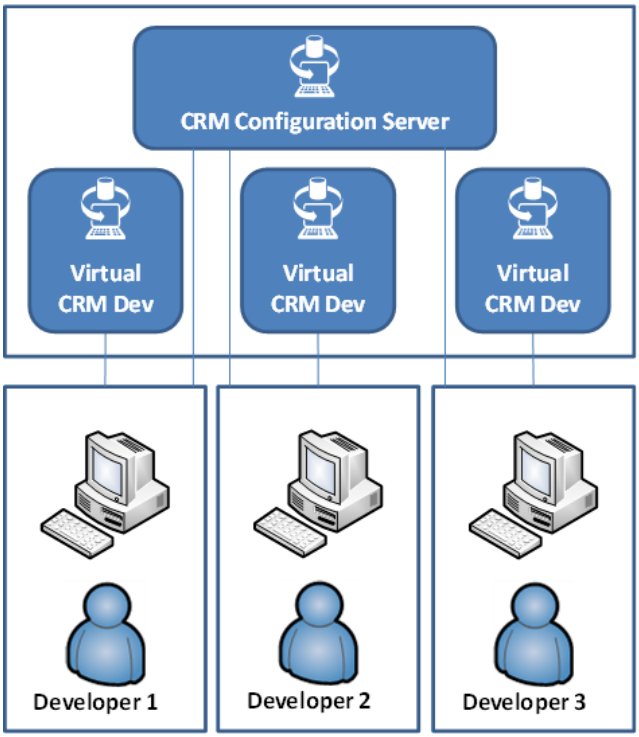


Multiple Virtual CRM Instances, Centrally or Locally Hosted

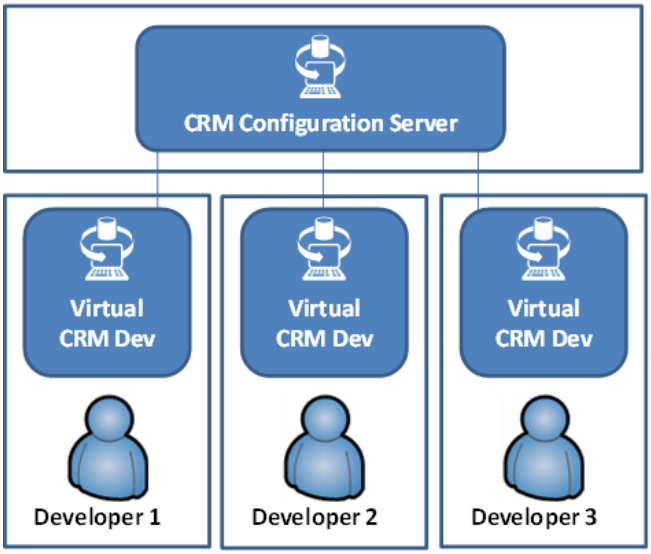
This scenario extends on the previous and enables the developer to have a completely isolated environment. Physically, this may be implemented on one or more centralized managed physical hosts if IT wishes to keep control over the infrastructure. Alternatively, developers may run a virtualization platform capable of supporting a Dynamics CRM 64-bit virtual machines (that is, Windows Server 2008 R2 with Hyper-V or Windows 8 with Hyper-V).

In this instance, a centrally managed deployment would still be required to host the central configuration organization. It is also recommended to provide a centrally managed virtual machine to each developer so that they have an identical platform and configuration.

Virtual CRM servers, centrally hosted with central configuration organization



Virtual CRM servers, locally hosted with central configuration organization



Pros

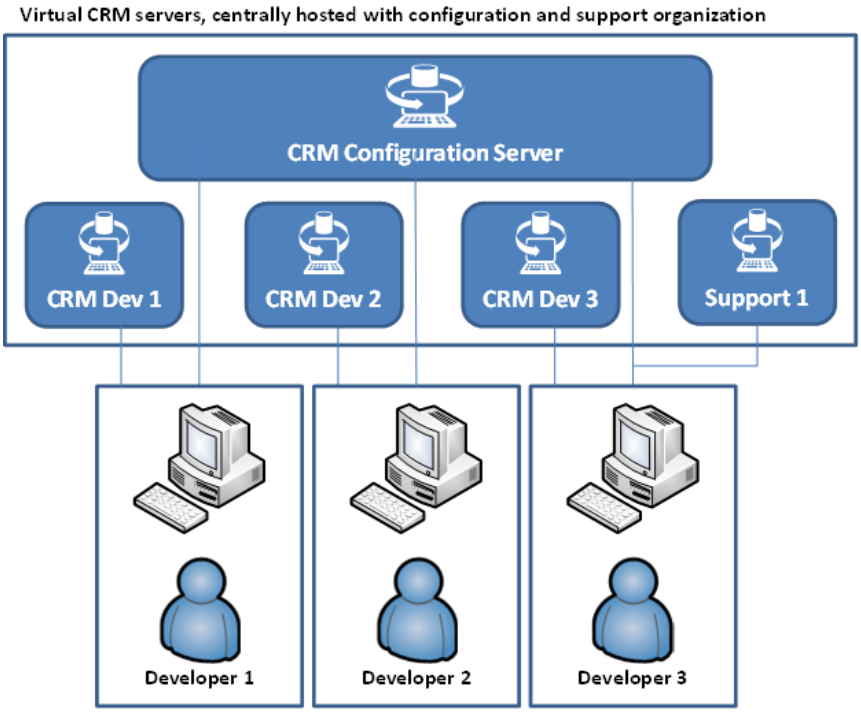
- Complete developer isolation.
- Fast “onboarding” of a new developer; just copy the virtual CRM development machine.
- Ability to snapshot the complete virtual machine and restore to a known state.
- Central configuration organization is kept clean from trial development work.
- Developer is free to create CRM organizations etc. for complex trials in his own environment.
- Developer has the exact development environment identified in the project specification, with no variations.
- Project team can store several states of the virtual CRM development machine for setting up or reverting to certain project phase or milestone specifics (for example, productive system is running on Update Rollup 3 level; V-Next development is based on latest Update Rollup package.
- May also already include project-specific CRM data like users, teams, business unit structure, and also basic CRM sample data.

Cons

- Infrastructure and licensing costs.
- All organizations would need to be at same Update Rollup level. If there are code changes as well as DB schema changes, this could exclude the “pro” to rollback an individual developer’s organization database.
- Process complexity; developers must export the CRM solution from their central configuration organization and import into their personal development organization to have all up-to-date configuration changes.
- Responsible person required for maintaining the dev master (if centrally managed).
- Effort for maintenance of centrally provided developer image.

Just as in the other development environments shown above for supporting simultaneous development of hotfixes, an additional CRM support organization for hotfix development may be required on the specific developer’s environment.

To support different Update Rollup versions in development and production environments, an additional environment for another CRM server may be required so the developers would need to switch over to a different virtual machine when working on another Update Rollup package level.



Multiple Development Teams