LE MAGAZINE DU DÉVELOPPEMENT

Programmez !

NUMÉRO SPÉCIAL

Microsoft® Visual Studio® Team System

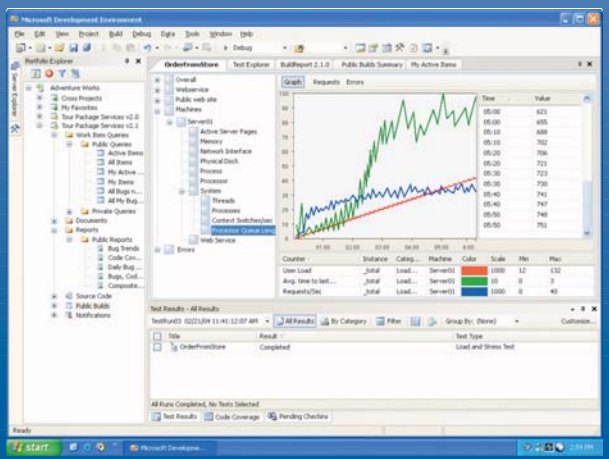
L'USINE à LOGICIELS de MICROSOFT

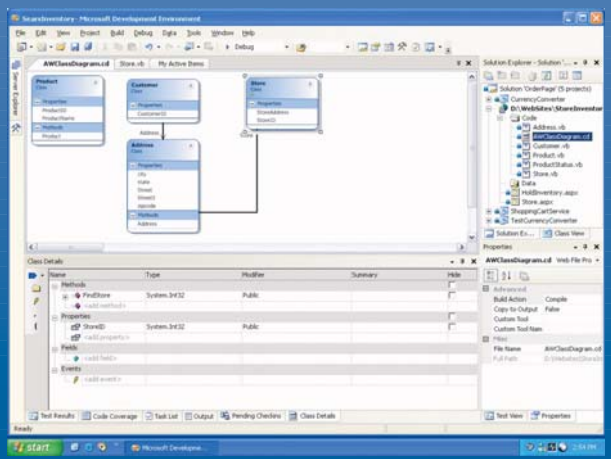
LA MODÉLISATION

UN ENVIRONNEMENT TRÈS INTÉGRÉ

DÉVELOPPEMENT PILOTÉ PAR LES TESTS

LES FONDATIONS D'UN SERVEUR DURABLE





Numéro spécial - Mars/Avril 2006

Mon usine à logiciels

Avec la complexité croissante des applications, des architectures, la nécessité de tenir les délais et de respecter les spécifications, la mise en place de solution de gestion de cycle de vie des applications se généralise, partiellement ou en totalité. Que l'on soit dans un « petit » ou dans un « grand » projet, le cycle de vie concerne tout le monde, et pas seulement les développeurs. Il suffit de piocher le bon composant, le bon outil.

Dans les environnements complets, les éditeurs prennent position. Microsoft, jusqu'à présent absent, laissait le développeur, le chef de projet, l'architecte choisir parmi les solutions d'éditeurs tiers. Désormais, avec la gamme Visual Studio Team System, Microsoft propose une solution intégrée de gestion cycle de vie orientée « rôle ». On y trouve des éditions pour l'architecte, le développeur, le testeur, et bien entendu, la partie serveur, la colonne vertébrale de l'ensemble qui répondra aussi aux besoins du chef de projet. Cette version 2005 constitue la nouveauté majeure de Visual Studio et de .NET 2.0. Désormais, il est possible de réaliser une modélisation d'architecture et de déploiement, définir les processus de tests, gérer au plus près les versions et les build grâce à des fonctions dédiées dans un environnement unique.

Cette première version démontre tout le potentiel de la gamme Team System. Même si par rapport aux solutions éprouvées du marché, on manque de recul et de retour d'expérience, Microsoft devient un acteur non négligeable du cycle de vie et l'intégration des outils à .NET et Visual Studio, un atout important. L'intégration n'est pas un vain mot. Le but est de rendre transparent le cycle de vie aux utilisateurs et que chaque « rôle » communique, sans se soucier du format, des protocoles.

Microsoft Visual Studio Team System

Pour une première version, Microsoft s'est concentré à bâtir un référentiel serveur, le cœur du système, et à fournir les outils périphériques majeurs du spectre fonctionnel. Il peut exister des lacunes sur certains domaines comme les tests fonctionnels ou la modélisation (pas d'outil UML 2.0), mais l'environnement peut être complété avec des solutions de partenaires comme Together ou Compuware. La version 2.0 de l'environnement, prévue pour la version de Visual Studio « Orcas » complètera les fonctionnalités.

Mais d'ores et déjà, nous pouvons dire que Microsoft s'impose comme un acteur du cycle de vie et de l'industrialisation du développement de logiciel. Il comble, en partie, son retard sur ses concurrents directs, même si la solution demeure centrée Windows et .NET. Mais le portage d'importants projets comme Ant ou Junit sur la plate-forme .NET démontre que .NET progresse et qu'il est une alternative visible et considérée, sur le marché, en particulier face à Java / J2EE. Et ces technologies s'intègrent très bien à Visual Studio... À vous de faire le choix !

François Tonic

sommaire

Programmez! LE MAGAZINE DU DÉVELOPPEMENT

Rédaction : redaction@programmez.com

Rédacteur en Chef : François Tonic.

Ont collaboré : A. Fontaine, D. Frontigny, F. Schäffer, C. Thibaut.

Maquette : PLB Communication.

Editeur : Go-02 sarl, 6 rue Bezout 75014 Paris
Dépôt légal : 1er trimestre 2006
Commission paritaire : 0707K78366
ISSN : 1627-0908
Imprimeur : ETC - 76198 Yvetot

Directeur de la publication : Jean-Claude Vaudecrane.

Abonnement : Programmez 22, rue René Boulanger, 75472 Paris Cedex 10 - abonnements.programmez@groupe-gli.com - Tél. : 01 55 56 70 55 - Fax : 01 55 56 70 20

La gamme Visual Studio Team System

Jusqu'à présent, Microsoft ne fournissait pas ou peu de choses autour du test, de la modélisation, du travail en équipe, de la gestion des sources. Avec Team System, l'éditeur tente une incursion dans un marché qui lui est inconnu. Bien qu'incomplet, l'environnement est prometteur. Reste à savoir si les développeurs et entreprises se laisseront séduire. L'éditeur propose donc une gestion de cycle de vie assez complète (appelé SDLC par Microsoft pour Software Development Life Cycle).

Orienté entreprise, Team System a pour objectif de fournir une « usine à logiciel », c'est-à- dire, réduire le temps de développement, automatiser un maximum de fonctions, avoir un environnement intégré, réaliser du code et des applications de qualité. Team System mise sur les bonnes pratiques, le travail en équipe, la gestion de configuration et de projet, ainsi que l'intégration continue et l'utilisation massive de métriques. Team System s'appuie sur MSF 4.0. MSF (Microsoft Solutions Framework) est un framework offrant les bonnes pratiques utilisées dans Team System. C'est ce que Microsoft nomme « méthode agile ». Par défaut, on dispose de MSF et MSF Agile. Mais comme le framework est extensible via un SDK et des API, il est possible d'implémenter sa propre méthode agile ! Foundation constitue la pièce maîtresse de l'ensemble. Il s'agit d'un référentiel centralisant l'ensemble des données et informations liées aux projets. Il utilise SQL Server 2005 pour la partie stockage.

La gamme Team System se découpe en trois éditions : Software Architect, Software Developers et Software Testers. Comme la concurrence, Microsoft a opté pour une approche rôle, chaque rôle ayant des besoins communs et spécifiques à son travail. À cela se rajoute Le Team Foundation Server, pour la partie serveur de l'environnement. Foundation est la tour de contrôle. Il stocke les projets, les métriques, les méthodes, les sources, la documentation. Surtout, il ne se limite pas à du classique client / serveur. Il propose un portail projet accessible depuis n'importe où ! L'outil est orienté services. Car on peut invoquer les services de Foundation via des Web Services. Il utilise WSS (Windows SharePoint Services) et des outils de reportings de projet pour suivre l'évolution de chaque tâche et du projet global.

Une segmentation bien pensée

Les différentes éditions incluent toutes : le désigner de classe, Visio, Team Explorer et l'édition professionnelle de Visual Studio 2005. Le Software Architect comporte spécifiquement les désigners d'application, système, de datacenter (partie logique) et de déploiement. Le Software Developers bénéficie des analyses de code statique et dynamique et d'un profileur de code. Il partage avec le Software Testers les tests unitaires et la couverture de code. Par contre, le Software Testers garde pour lui les fonctions de tests de charges, les tests manuels et le Test Case Management. Si vous souhaitez utiliser l'ensemble des fonctions des différents rôles, vous pouvez acquérir Team Suite qui inclut les éditions architecte, développeur et testeur. De plus, chaque version de Team System inclut en standard les outils pour le développement Office (VSTO). À noter que pour le Software Testers, il est possible de le coupler à la solution de virtualisation de Microsoft : Virtual Server 2005.

Team Foundation Server est la partie serveur de Team System. Pour fonctionner, il faut acquérir des licences d'accès client (ou CAL). Chaque client accède au serveur. Il est possible d'utiliser du Visual Studio 2003 en environnement Team System, sans les expériences utilisateurs de la gamme 2005. Basiquement, Foundation Server permet de créer et de gérer des projets, des tâches (besoins, bugs...), le versioning, de générer des rapports, d'automatiser les builds. L'outil s'appuie sur SQL Server Reporting Service pour tout ce qui est reporting, et sur Windows SharePoint Services pour le portail projet.

Les tarifs sont les suivants (uniquement pour 1 à 4 licences, avec abonnement MSDN Premium) :
Software Architects : 5 899 €
Software Developers : 5 899 €
Software Testers : 5 899 €
Team System Suite : 11 711 €

LES PRÉVISIONS SUR LA VERSION 2.0

Team Architect v1	Team Architect v1	Team Test v1
support WSE support Click-once Designer de déploiement Moteur de validateur Designer d'application	support analyseur code statique (code managé, non managé) profiler de code code coverage tests unitaires (similaire à Nunit)	test de charge scénario de tests
Team Architect v2	**Team Architect v2**	**Team Test v2**
support d'Indigo Designer Business Process Moteur de pattern	profiling au niveau système et monitoring d'application amélioration de l'analyse statique	test fonctionnel (Avalon, WinForms, Win32, IE) gestion Test Lab historique du code coverage

François Tonic

Team System : l'intégration entre IDE et outils de gestion

Comment se positionne Team System ? Quels arguments Microsoft donne-t-il pour convaincre développeurs et entreprises d'utiliser Team System ? Antoine Driard, spécialiste Solutions de Développement Microsoft France, s'est prêté au jeu des questions – réponses.

Programmez ! : *C'est la première fois que Microsoft investit sur les outils de cycle de vie, les outils de tests. Quelles sont les attentes, les espoirs de l'éditeur par rapport à Team System et Team System Foundation Server ?*
Antoine Driard : Notre ambition principale est de mettre à la portée des équipes de développement des outils jusque-là essentiellement réservés aux grosses équipes à cause de leur complexité, de leur coût, par exemple dans les environnements de gestion de changement ou de configuration. Nous souhaitons démocratiser ces outils, à un coût relativement accessible. Nos clients, les utilisateurs, voulaient aller au-delà de l'environnement de développement.

Programmez ! : *Depuis quelques mois, des projets indépendants voient le jour, comme celui de Devbiz, avec le plug-in Eclipse TeamPlain, permettant de se connecter à un Team System Foundation Server à partir d'un IDE autre que Visual Studio. Microsoft encourage-t-il ce genre d'initiative ?*
Antoine Driard : Il n'y a pas de grande équipe utilisant uniquement les outils et technologies Microsoft. Il y a un besoin d'intégration d'outils, de langages non Microsoft. Nos outils de développement ciblent uniquement Windows, .Net et le C++ natif. Mais nous avons des clients développant en Java sur Windows ou Linux qui ont vu en Team Foundation un environnement répondant à leur attente pour ses fonctionnalités serveur (gestion de configuration, suivi de bugs...). On s'attendait donc à ce qu'il existe des projets tiers pour connecter d'autres outils à Team Foundation. Toutes les fonctions de Team Foundation Server sont publiées en tant que services web, facilitant leur intégration dans d'autres environnements.

Programmez ! : *Vous mettez en avant l'aspect intégration, est-ce un élément important ?*
Antoine Driard : Certainement. La forte intégration entre Team System et Visual Studio dégage une valeur ajoutée, on réduit le cycle de formation et parfois il n'y a pas de formation du tout. Ce n'est pas toujours le cas quand on utilise un environnement de développement avec des outils tiers hétérogènes.

Programmez ! : *On a souvent glosé sur l'absence d'UML dans Team System. Pourquoi avoir choisi une approche « éclatée » de la modélisation ?*
Antoine Driard : Nous avons une approche différente d'UML. Même si nous n'investissons pas sur UML, nous ne le remettons pas en cause. Si vous souhaitez faire de la modélisation UML il existe des outils de modélisation sur le marché et sur ce point, nous nous reposons sur nos partenaires. UML considère que tout peut être modélisé, pris en compte dans un unique modèle. Nous, nous pensons que plusieurs modèles spécifiques seront plus précis, plus pertinents, moins génériques. Grâce à cela, nous sommes capables dans Team System de modéliser de l'infrastructure matérielle au déploiement de l'application. On modélise les différentes couches de l'environnement applicatif. Même si les différents modèles n'interagissent pas à 100 % ensemble, il y a toujours des éléments sous-jacents communs entre tous les modèles que ce soit sur le code ou XML. De plus, nous livrons des SDK permettant à chaque utilisateur de créer son propre modèle adapté rigoureusement à ses besoins.

Programmez ! : *Microsoft a longtemps parlé de méthodes agiles dans le développement avec Visual Studio et encore plus sous Team System. Comment cela se traduit-il ?*
Antoine Driard : Effectivement. Nous livrons en standard 2 méthodes : MSF Agile et MSF CMMi. L'idée est de fournir des templates de méthodologie définissant la gestion du code, les tests, les processus de développement, etc. Il s'agit donc d'automatiser les différentes tâches et les interactions des intervenants du projet. Au-delà de ces deux templates, d'autres méthodes plus simples verront le jour à la sortie du produit. Il y a deux niveaux d'utilisation de ces templates. Le premier est l'utilisation telle quelle de la méthode notamment dans les petites équipes de développements. Le second est la personnalisation de la méthode, plus sensible dans les grosses équipes possédant souvent leurs propres processus.

Programmez ! : *Faut-il s'attendre pour cette année 2006, à des événements spécifiques Team System de la part de Microsoft ?*
Antoine Driard : Oui ! Il y a déjà une session orientée tests aux DevDays et des événements moins grand public pour les architectes. Même si nous n'avons pas encore de calendrier précis, il y a des événements spécifiques Team System purement Microsoft ou co-animés par nous.

Propos recueillis par François Tonic

Visual Studio Team System : l'environnement très intégré

La version 2005 de Visual Studio est l'une des plus novatrices, puisqu'elle dépasse le monde de la conception logicielle pure pour s'étendre à la quasi-totalité du cycle de vie du développement, avec l'extension Team System.

Team System est composé de deux blocs : un ensemble de fonctionnalités intégrées à Visual Studio 2005 (VS 2005), et une couche serveur, Team Foundation Server (TFS). L'ensemble se concentre sur les spécificités du développement en équipe, et se traduit par des outils de contrôle de source, de qualimétrie, et d'aide à la méthodologie.

Toutes les données associées à ces outils sont stockées et centralisées dans SQL Server 2005. Cette façon de procéder permet à Microsoft d'intégrer chaque brique avec les autres et de proposer un ensemble de Web Services que n'importe quelle application tierce peut utiliser pour consulter et enrichir TFS. Cela signifie qu'il n'est pas nécessaire d'avoir VS 2005 pour travailler avec TFS. Microsoft a en effet exploité ces web services dans un client autonome (Team Explorer), mais également dans VS 2003 (notamment pour le contrôle de sources).

Le contrôleur de Sources

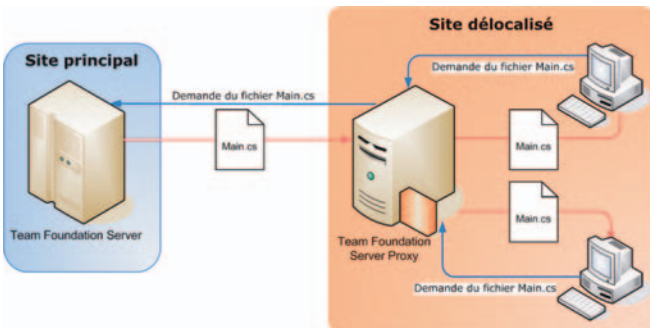
À partir de VS 2005, Microsoft propose deux contrôleurs de source :
- Microsoft Visual SourceSafe 2005.
- TFS Version Control (TFS-VC).

Le premier est destiné au développeur isolé ou aux petites équipes. Microsoft place la barre à 5 utilisateurs, mais le critère déterminant est plutôt le nombre de personnes travaillant simultanément sur un même projet, ainsi que la délocalisation physique. Visual SourceSafe 2005 est utilisable sans Team System, et comporte peu d'améliorations.

TFS-VC est tourné vers les gros projets, et apporte un ensemble de notions indispensables au développement en équipes :

Proxy

Si votre équipe est délocalisée sur plusieurs sites physiques, la bande passante dont vous disposez est un facteur limitant dans l'organisation des développements : un outil classique de gestion de sources demande à chaque développeur de fréquemment se resynchroniser avec le référentiel des sources, et cette opération peut devenir très longue lorsque les projets grossissent. TFS-VC propose des proxies installables localement pour mutualiser les demandes de mise à jour et réduire ainsi le temps de resynchronisation.



Workspaces

Un Workspace est l'ensemble des fichiers utilisés par un développeur sur sa machine, que ces fichiers soient extraits en lecture seule ou pour modification. TFS-VC permet de récupérer le Workspace sur une autre machine, ou bien de récupérer le Workspace d'un autre développeur. Cette fonctionnalité va jouer un rôle fondamental dans les applications composées d'un très grand nombre de projets, eux-mêmes mutualisés entre plusieurs applications. En effet, le travail fait par un développeur consistant à extraire sous une arborescence locale uniquement les projets intéressants est partageable avec les autres développeurs, et automatisable pour recréer un nouvel environnement de travail lorsqu'un nouveau développeur rejoint l'équipe.

ChangeSet

Nous sommes vendredi, il est 17h00, et vous avez passé les deux derniers jours à effectuer des modifications. Tout compile enfin, et vous faites un check-in pour réintégrer vos modifications dans l'arbre des sources. Sur les 80 fichiers que vous venez de mettre à jour, 74 se passent bien, et 6 comportent des conflits. Avant TFS-VC, vous ne pouviez pas annuler votre check-in : pendant que vous faisiez votre check-in, 8 autres développeurs ont fait le leur, et vous ne pouvez pas supprimer vos modifications sans annuler les leurs. Le ChangeSet est le nouveau moyen de gérer des mises à jour transactionnelles, là où VSS les traitait fichier par fichier. De cette façon, l'intégralité des modifications apportées au contrôleur de sources est, soit appliquée, soit annulée.

Shelving

Le shelving permet de réintégrer vos modifications sans pour autant les appliquer sur le tronc principal. Il s'agit d'une mini branche, qui vous est propre, et sur laquelle vous pouvez revenir plus tard.

Cette fonctionnalité est beaucoup plus simple de mise en œuvre que la création d'une branche, et elle permet de répondre aux scénarios suivants :
- Je n'ai pas le temps de terminer ce soir, mais je dois relâcher les fichiers.
- Je vais devoir continuer le travail depuis un autre poste.
- J'aimerais avant de réintégrer le tronc commun que quelqu'un jette un œil à mes sources.

Les outils de mesure de qualité logicielle

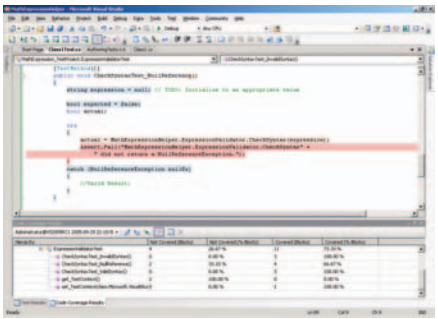
Analyse Statique

Il s'agit de l'intégration de FxCop dans Visual Studio. Le chef de projet peut définir un

ensemble de règles à surveiller et connaître immédiatement l'ensemble des violations. Il peut ensuite en faire une métrique, ou être plus restrictif et mettre des barrières de check-in.

Couverture de code

Cette mesure s'applique sur les scénarios de tests et permet de distinguer les portions de code parcourues au moins une fois, lors de l'exécution du programme, de celles qui n'ont jamais été balayées. Cet indicateur permet de s'assurer que les développeurs ne cessent pas de tester leur code en période de recette, par exemple.



Tests et organisation des tests

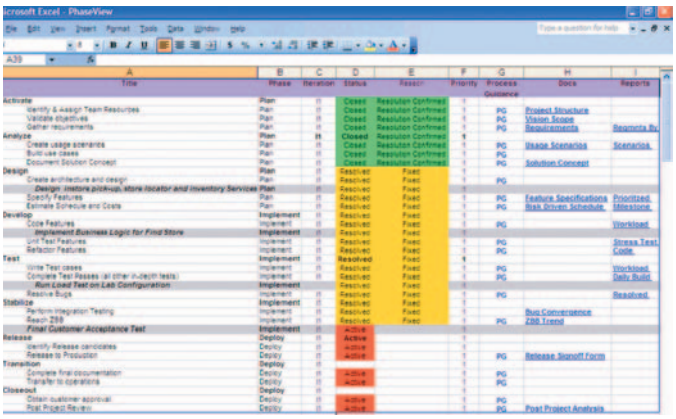
Vous avez la possibilité de créer de nouveaux tests en les associant à des tâches (anomalies, demandes d'évolutions, contraintes de performance). L'exécution des tests se fait ensuite selon deux grandes approches : les développeurs peuvent créer des regroupements de tests à l'aide d'un requêteur embarqué dans Visual Studio et les jouer sur la machine locale. Les testeurs peuvent, quant à eux, créer des scénarios complets qui seront grâce à Team Build joués sur des machines dédiées.

Les outils d'aide à la méthodologie

Barrières de Check-In

Cette possibilité a fait l'objet d'un débat chez Microsoft et a changé beaucoup d'habitudes de travail. Elle permet de définir des règles de réintégration des sources. Il arrive souvent que des développeurs réintègrent dans le contrôleur de source, du code qui ne compile pas. Cette pratique est inacceptable dans un projet en équipe, puisqu'elle empêche les autres développeurs de compiler. TFS-VC peut, si vous le paramétrez, bloquer le check-in tant qu'il n'est pas précédé d'une compilation sans erreur.

Les barrières de check-in disponibles par défaut :
- Clean Build (Build valide avant check-in, pour garantir que le code compile)
- Code Analysis (règles FxCop, pour garantir que les règles ont été observées)
- Testing Policy (pour garantir que le code a été testé)
- Work Item (pour garantir que les modifications correspondent à une demande)



Automatisation du processus de build

Team System dispose d'un serveur de build nommé Team Build, que vous pouvez installer sur n'importe quelle machine du réseau. Lorsque vous lancez un build, vous avez la possibilité de désigner une autre machine que la vôtre pour exécuter le build. Team Build est capable de récupérer une version spécifique des sources, de lancer les tests unitaires associés ou encore, de faire de l'analyse statique du code généré. Les résultats des builds sont ensuite archivés et permettent de générer des états graphiques d'avancement de projet.

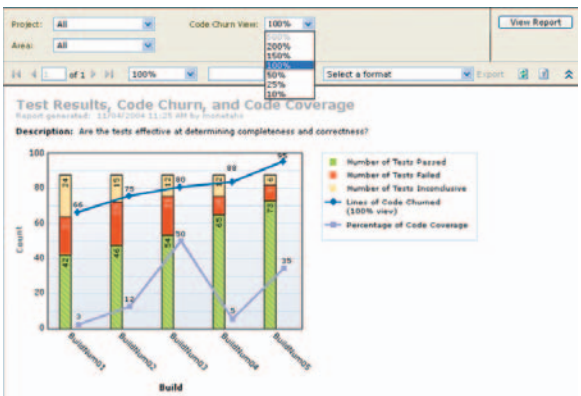
Work Items

Les Work Items représentent des tâches élémentaires. Ces tâches peuvent concerner toute personne associée au projet. Chaque tâche peut en entraîner une autre, et Team System possède un workflow sophistiqué, permettant à chaque membre d'une équipe projet de consulter et enrichir des Work Items depuis son environnement de travail habituel.

Imaginez, par exemple, que le responsable fonctionnel identifie un nouveau scénario d'utilisation de l'application. Ce scénario sera saisi de façon assez informelle dans un site Sharepoint. Un chef de projet va ensuite en déduire des tâches plus techniques qu'il affectera à des développeurs (par exemple depuis Project), ainsi que des tâches de test. Les développeurs et les testeurs vont ensuite travailler sur ces tâches techniques, signaler et corriger des bugs (depuis Visual Studio). L'évolution de la nouvelle fonctionnalité est consultable par le client dans une feuille Excel, où il pourra également valider les modifications apportées.

La liste des Work Items disponibles, ainsi que leurs enchaînements possibles, sont paramétrables, et Microsoft livre deux pré-paramétrages avec Team System. Le premier, nommé MSF Agile, est l'incarnation d'une méthodologie assez légère, inspirée des approches récentes (MSF, Xtreme Programming). Le second est plus formel et se destine aux équipes désireuses de se mettre en conformité avec CMMI.

Il est très simple de créer des états d'avancement projet mettant en lumière chaque aspect du développement, et Team System propose des tableaux de bord pour Sharepoint et Excel. Bien évidemment, vous pouvez également créer vos propres états ou explorer interactivement n'importe quelle métrique projet.



■ Davy Frontigny et Pierre Couzy
Winwise – www.winwise.fr

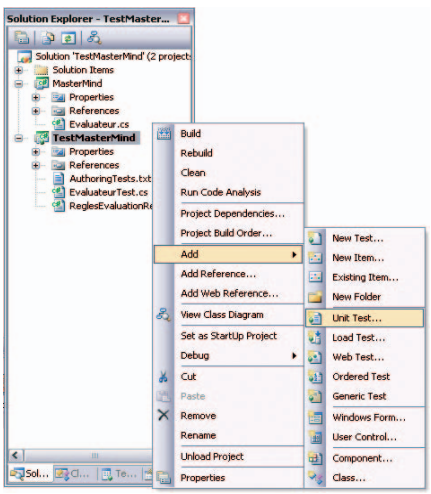
Développement piloté par les tests avec Visual Studio 2005 TeamSystem

L'environnement de développement de Microsoft apporte de nouveaux outils permettant de tester nos applications. Dans cet article, nous explorons certains de ces outils dans une démarche strictement TDD (Test Driven Development) : utilisation d'un cycle très court Test/Code/Refactor, couverture totale du code (chaque ligne de code est justifiée par un test qui ne passait pas) et design émergent.

Le cahier des charges

Nous souhaitons écrire un programme permettant de jouer au Mastermind. Au cours de la partie, le programme détient un code secret constitué de pions de différentes couleurs, que le joueur essaye de deviner en soumettant des propositions. Le programme évalue chaque proposition et renvoie deux indicateurs :
nombre de pions bien placés (même couleur et position que dans le code secret)
nombre de pions mal placés (même couleur à une position différente)
Pour décrire simplement et efficacement le comportement attendu du programme, nous constituons des tests de recette, sous forme tabulaire. Voici le test de recette dans lequel sont exprimées les règles d'évaluation *(cf tableau ci-dessous).*
Notre objectif étant d'écrire un jeu respectant ces règles, ce test de recette fournit une *liste de courses* idéale pour commencer à écrire le coeur du programme.

Préparation



Nous créons un projet de type « librairie de classes ». Pour écrire un test, il faut une classe de tests. VSTS peut créer cette classe pour nous, si nous lui indiquons la classe testée. Comme notre intention est d'écrire un programme d'abord capable d'évaluer une proposition, nous créons une classe *Evaluateur*, vide. Ensuite, en un clic sur l'option « Créer un projet de test » du menu contextuel, nous créons automatiquement un projet de test dans lequel figure une nouvelle classe *EvaluateurTest.*

Premier test

Nous commençons par la recherche des bien placés. Le premier test est trivial : il exprime le fait que si la proposition ne contient aucun pion, le nombre de pions bien placés est 0.

```
using
Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Text;
using System.Collections.Generic;
using MasterMind;

namespace TestMasterMind
{
  [TestClass()]
  public class EvaluateurTest
  {
    /*Code généré*/

    [TestMethod()]
    public void
PropositionVideProduit0BienPlaces()
    {
      Evaluateur evaluateur = new Evaluateur();
      String proposition = "";
      String codeSecret = "RVBNA";
      Assert.AreEqual(0,
evaluateur.EvalueBienPlaces(codeSecret,
proposition));
    }
  }
}
```

Règles Evaluation

codeSecret	proposition	bienPlaces	malPlaces	commentaire
RBBJA		0	0	proposition vide
RRRRR	R	1	0	un bien placé
RVVVV	V	0	1	un mal placé
VBBBB	BR	0	1	un mal placé et un non présent
RVBJA	RRR	1	0	ne compter qu'une fois le bien placé
RVVJA	RVVJA	5	0	combinaison trouvée
RRRRR	V	0	0	aucun trouvé
RVBNN	RVB	3	0	plusieurs bien placés
RVVVV	VVV	2	1	ne compter qu'une fois le mal placé parmi des bien placés
VVVVR	RRR	0	1	ne compter qu'une fois le mal placé
RVBAA	NRVBN	0	3	3 mal placés
BVRRR	RRJJN	0	2	2 identiques mal placées

Ce premier test obéit à la règle des trois A : pour écrire un test, définissez des Acteurs, lancez une Action puis faites une Assertion.

Ici l'acteur est un objet de la classe *Evaluateur*, l'action consiste à invoquer la fonction *EvalueBienPlaces*, et l'assertion *Assert.AreEqual* vérifie que le résultat de cet appel est bien 0.
Les attributs *TestClass* et *TestMethod* présents dans le source indiquent à VSTS que ce code est constitutif des tests du projet.
En TDD nous écrivons le test avant d'écrire le code qui passe le test. Ici, Intellisense ne nous aide pas vraiment : comme le code à tester n'existe pas encore, les suggestions de complétion sont inappropriées, ce qui oblige à les ignorer. VSTS se rattrape un peu tout de même, en proposant via le clic droit, de créer les squelettes des méthodes qui n'existent pas encore. Notons que si VSTS permet de générer des accesseurs spécifiques pour tester les membres privés de classe, nous n'utiliserons pas cette fonctionnalité. En effet, le fait d'avoir à tester les membres privés d'une classe indique un problème de design, qu'il faut résoudre en scindant la classe et en découplant le code.

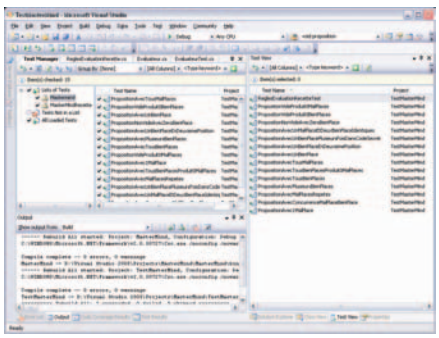
Une fois que le code est syntaxiquement correct, nous faisons en sorte que le test échoue lors de sa première exécution. Nous faisons cela en forçant la valeur de retour de la fonction testée :

```
public int EvalueBienPlaces(String codeSecret,
String proposition)
{
  return -1;
}
```

Nous pouvons maintenant exécuter le test : nous ouvrons la fenêtre « Test View » depuis le menu Test/Windows. Grâce aux attributs *TestClass* et *TestMethod*, VSTS a pu recenser notre premier test et l'inclure dans la liste. Nous le sélectionnons et l'exécutons. Conformément à nos attentes, le test échoue. Maintenant, nous pouvons modifier le code pour faire en sorte qu'il passe. La solution la plus simple est de changer la valeur de retour :

```
public int EvalueBienPlaces(String codeSecret,
String proposition)
{
  return 0;
}
```

Et maintenant le test passe (icône verte). Cette façon de procéder répond au pattern Error, Failure, Success : d'abord le code ne compile pas ; ensuite on s'assure que le test échoue ; enfin on écrit le code le plus simple qui fasse passer le test.



Deuxième test

Le test suivant consiste à s'assurer que l'évaluateur détecte correctement une couleur bien placée dans la proposition. Afin de localiser plus facilement nos erreurs, nous choisissons de créer un nouveau test pour chaque nouvelle assertion.

```
[TestMethod()]
public void PropositionAvecUnBienPlace()
{
  Evaluateur evaluateur = new Evaluateur();
  String proposition = "R";
  String codeSecret = "RVBNA";
  Assert.AreEqual(1, evaluateur.EvalueBien
Places(codeSecret, proposition));
}
```

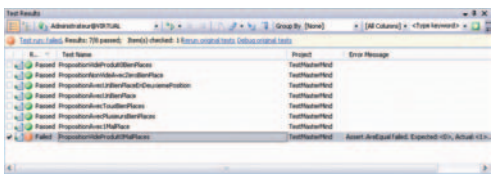
Nous modifions notre code de production, afin de faire passer ce test. Nous constatons ensuite qu'il est possible de factoriser le code de test, en faisant de la variable *evaluateur* un champ privé de la classe de test. Ce champ pourra être initialisé avant chaque test par une méthode *SetUp*, bien connue des utilisateurs de xUnit. Pour être invoquée avant chaque test, cette méthode doit être munie d'un attribut *TestInitialize* :

```
private Evaluateur evaluateur;

[TestInitialize()]
public void SetUp()
{
  evaluateur = new Evaluateur();
}
```

```
[TestMethod()]
public void Proposition
VideProduit0BienPlaces()
{
  String proposition = "";
  String codeSecret = "RVBJA";
  Assert.AreEqual(0, evaluateur.EvalueBien
Places(codeSecret, proposition));
}

[TestMethod()]
public void PropositionAvecUnBienPlace()
{
  String codeSecret = "RRRRR";
  String proposition = "R";
  Assert.AreEqual(1, evaluateur.
EvalueBienPlaces(codeSecret, proposition));
}
```

Afin de vérifier que ce refactoring n'a pas provoqué d'erreur, nous repassons les tests. Ils sont tous à l'état vert. Nous venons de modifier le code de test en utilisant le code de production comme filet de non-régression.
Règle : le code de test doit évoluer parallèlement au code de production.

Tests suivants



La fonction d'évaluation des bien placés évolue test après test, pour aboutir au code suivant (nous supposons que les paramètres passés seront contrôlés par la classe utilisatrice) :

```
public int EvalueBienPlaces(String codeSecret,
String proposition)
{
  int bienPlaces = 0;
  for (int i = 0; i < proposition.Length; i++)
    if (proposition[i] == codeSecret[i])
      bienPlaces++;
  return bienPlaces;
}
```

Nous passons à l'évaluation des mal placés. Notre premier test exprime le fait que lorsque la proposition est vide, le nombre de mal placés est forcément 0. Pour passer ce test, la fonction *EvalueMalPlaces* renverra 0. (La répon-

se la plus simple qui fasse passer le test). Ensuite nous amenons un nouveau test : avec un code secret égal à « RVVVV », et la proposition « V », *EvalueMalPlaces* doit renvoyer 1. Nous pensons pouvoir faire passer ce nouveau test, ainsi que le précédent en écrivant ceci :

```
public int EvalueMalPlaces(String codeSecret,
String proposition)
{
  if (codeSecret.Contains(proposition))
    return 1;
  else
    return 0;
}
```

Mais le test précédent casse, car *Contains* avec une chaîne vide en argument renvoie toujours true. *Pour détecter au plus tôt toute régression, il faut exécuter tous les tests et non seulement le dernier.*
Au fil des tests et des refactorings, nous obtenons un code dans lequel nous avons confiance et que nous jugeons capable de satisfaire les tests de recette :

```
public int EvalueMalPlaces(String codeSecret,
String proposition)
{
  int malPlaces = 0;
  bool []dejaCompte = new
bool[codeSecret.Length];

  for (int i = 0; i < proposition.Length; i++)
  {
    int positionDansCodeSecret =
codeSecret.IndexOf(proposition[i]);
    if ((positionDansCodeSecret != -1) &&
      (proposition[i] != codeSecret[i]) &&
      !dejaCompte[positionDansCodeSecret])
    {
      malPlaces++;
      dejaCompte[positionDansCodeSecret] =
true;
    }
  }

  return malPlaces;
}
```

Tests fonctionnels

Jusqu'ici, le tableau initial concernant les règles d'évaluation nous a servi de liste de courses pour la programmation TDD. Nous voudrions maintenant définir un test fonctionnel capable de valider le code à un niveau plus élevé.
Nous pourrions écrire un test de recette automatisé dans lequel tous les cas seraient examinés, c'est-à-dire tous ceux auxquels le testeur aura pensé jusqu'ici. Pour ajouter de nouveaux cas, le testeur aurait besoin de modifier ou faire modifier le code du test, ce qui est un problème.

Test manuel

A l'inverse, nous pourrions également créer un « test manuel » et le conserver dans VSTS. Ce type de test consiste à afficher un document Word décrivant le mode opératoire du test et que l'utilisateur peut valider manuellement via une case à cocher. Mais les tests manuels sont fastidieux et sujets à erreur, et ils nous obligeraient à faire dépendre d'une interface utilisateur pour tester le cœur de notre programme. Comme nous voulons, dans la mesure du possible, automatiser les tests de recette, nous mettons en place une solution consistant à interpréter une feuille Excel contenant les cas de test : nous allons « détourner » un test unitaire et le doter d'une liaison avec le fichier Excel contenant les règles d'évaluation citées en début d'article.

Dans la feuille Excel, nous définissons la zone de données (y compris les en-têtes des colonnes) et la baptisons « ReglesEvaluation ». Ensuite, nous créons une nouvelle classe *MasterMindRecette*. Nous ne pouvons pas la créer en tant que classe de test à l'aide de l'assistant VSTS, qui ne crée des classes de tests qu'à partir de classe de code de production existante. Décidément, Visual Studio n'est pas tout à fait « TDD-Ready »... Il faut donc ajouter l'attribut *TestClass* à cette classe. Puis, nous créons une propriété de type *TestContext*, qui va permettre à VSTS de nous donner le contexte de ce test un peu spécial.
Un attribut *DataSource*, paramétré avec les caractéristiques de la source de données utilisée (pilote, chaîne de connexion, nom de la source de données, et type d'accès), va permettre au code de test de retrouver ses petits...

```
[DataSource("System.Data.Odbc",
"Driver={Microsoft Excel Driver
(*.xls)};DriverId=790;Dbq=C:\\TMP\\
TestMasterMind\\MasterMind.xls;DefaultDir=
C:\\TMP\\TestMasterMind\\;",
"ReglesEvaluation",
DataAccessMethod.Sequential), Timeout(0),
TestMethod]
```

L'accès à une cellule de la feuille se fait par l'attribut DataRow*[« nom de colonne »]*. Dans le test, nous lisons les cellules correspondant au code secret, à la proposition, au nombre de bien placés attendu et au nombre de mal placés attendu.

```
public void ReglesEvaluationRecetteTest()
{
  string codeSecret = TestContext.
DataRow["codeSecret"].ToString();
  string proposition = TestContext.
DataRow["proposition"].ToString();
  int bienPlaces_attendu = Int32.Parse(Test
Context.DataRow["bienPlaces"].ToString());
  int malPlaces_attendu = Int32.Parse
(TestContext.DataRow["malPlaces"].ToString());
```

Ces informations vont permettre d'appeler l'évaluateur et de comparer les nombres attendus aux nombres obtenus. Au passage, nous écrivons sur la console une trace du cas de test en cours d'exécution, de manière à le repérer plus facilement en cas d'échec. Cette ligne sera insérée par VS dans le rapport de test.

```
  Evaluateur evaluateur = new Evaluateur();

  int bienPlaces_effectif = evaluateur.EvalueBien
Places(codeSecret, proposition);
  int malPlaces_effectif = evaluateur.EvalueMal
Places(codeSecret, proposition);

  Console.WriteLine("Valeurs d'entrée : code
Secret : {0}\t proposition: {1}",
        codeSecret, proposition);
  Assert.AreEqual(bienPlaces_attendu,
bienPlaces_effectif,
        " bienPlaces_attendu was not set cor
rectly.");
  Assert.AreEqual(malPlaces_attendu,
malPlaces_effectif,
        " malPlaces_attendu was not set
correctly.");
}
```

Lors de sa première exécution, le test de recette révèle une erreur de programmation, que nous n'avions pas détectée en tests unitaires. Nous créons donc un nouveau test unitaire comportant le cas identifié, puis nous corrigeons le code. C'est une pratique recommandée en TDD de mettre en évidence un bug à l'aide d'un test, puis de laisser le test dans la base de code afin que le bug ne morde pas deux fois.

Au final, notre code de production est le suivant :

```
public int EvalueMalPlaces(String codeSecret,
String proposition)
{
  int malPlaces = 0;
  bool[] dejaCompte = new bool
[codeSecret.Length];

  for (int i = 0; i < proposition.Length; i++)
  {
    int positionDansCodeSecret = -1;
    int positionDebutRecherche = 0;
    bool bFinRecherche = false;

    while (bFinRecherche == false)
    {
      positionDansCodeSecret = codeSecret.
IndexOf(proposition[i],positionDebutRecherche);
      if (positionDansCodeSecret == -1)
        bFinRecherche = true;
      else
      if (!dejaCompte[positionDansCodeSecret])
          bFinRecherche = true;
        else
          positionDebutRecherche =
positionDansCodeSecret+1;
    }

    if (positionDansCodeSecret != -1)
    {
      dejaCompte[positionDansCodeSecret] =
true;
      if (proposition[i] != codeSecret[i])
        malPlaces++;
    }
  }

  return malPlaces;
}
```

Comme nous ne sommes pas tout à fait satisfaits du design, nous tentons de le réarranger, notamment en factorisant la recherche des éléments bien et mal placés dans une seule méthode. Mal nous en prend : deux tests cassent ; aussi nous limitons nos prétentions et essayons de refactorer par plus petites étapes. Nous n'avons aucune crainte à modifier ainsi le code, puisque nous pouvons en vérifier le comportement à tout moment. La stratégie TDD ne garantit pas un code « parfait » d'entrée de jeu, mais elle nous garantit un code améliorable en toute confiance.

Test de couverture

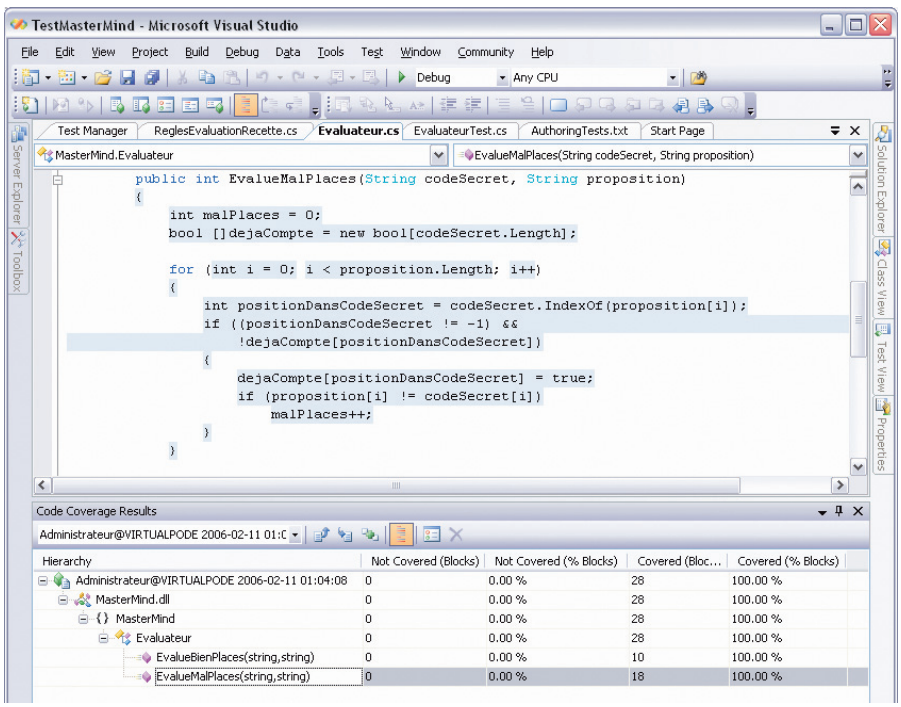
Comme nous savons que VSTS dispose d'un outil de mesure de la couverture de test, nous appliquons sur la solution l'option de configuration « code coverage ».
La couverture de test du code est évidemment de 100%, puisque chaque ligne de code est justifiée par un test qui ne passait pas. Lorsqu'on pratique TDD de manière systématique, cet outil devient relativement superflu. En revanche, appliqué à un code trop peu testé, il permet sans aucun doute de souligner la douleur…

Pour aller plus loin

Un test de recette passe sur notre code. Cela ne signifie pas que notre programme soit terminé, mais il est en bonne voie ! Lorsqu'une interface utilisateur sera mise en place, celle-ci pourra être testée fonctionnellement via les tests Web (analyse des réponses aux requêtes HTTP). VSTS fournit également des outils de montée en charge, peu utiles dans le cadre de notre exemple.

- conception évolutive facilitée (les tests effacent la peur de réaménager le code) ;
- confiance accrue dans le code.

L'environnement fournit un outillage complet autour des tests : unitaires, fonctionnels, pilotés par les données, de couverture, de montée en charge. La possibilité de lire directement les documents au format Excel, maîtrisé par le client responsable de la recette, est certainement un atout. Nous regretterons tout de même la difficulté de mise en oeuvre et le fait que le résultat des tests ne soit pas réintégrable simplement dans les pages de tests Excel (comme le propose le framework FIT par exemple).
L'EDI n'encourage pas, même s'il ne l'empêche pas complètement, le principe TDD d'écrire chaque test avant le code, ce qui est dommage. En revanche, les outils de refactoring et de génération de squelettes de méthodes fournis par VSTS assistent correctement la démarche. Il est tout à fait possible d'écrire des tests unitaires dans la plus pure tradition xUnit, aux mots-clés près.



Conclusion

Il est parfaitement possible d'adopter la démarche Test Driven Development dans l'environnement Team System. Nous profiterons ainsi des avantages de cette méthode :

- temps de mise au point réduit (le « boomerang » revient plus vite ; les défauts sont détectés très rapidement) ;

Christophe THIBAUT, Architecte senior
cthibaut@octo.com
Frédéric SCHÄFER, expert - fschafer@octo.com

Fondé en 1998, OCTO Technology est le premier cabinet d'architectes des systèmes d'information.
www.octo.com

Team System et la modélisation

La gamme Team System est, comme nous l'avons déjà dit dans les pages précédentes, orientée rôles, elle couvre un large spectre du cycle de vie de l'application et reprend le concept d'usines à logiciels. La modélisation et l'agilité font partie de cette définition. Dans les pages qui suivent, nous aborderons principalement le problème de la modélisation, des choix de Microsoft, dans quels cadres la modélisation intervient dans Team System et quelle est son utilisation dans les projets .Net 2.

Basiquement, Team System possède 4 designers de modélisation : pour l'application, pour le système, pour l'infrastructure technique (= data center) et pour le déploiement. Dans la gamme Team System, il y a des différences. Si toutes les éditions ont le designer de classes, seule l'édition architecture possède les designers de datacenter, de déploiement et d'application. Visio qui propose une modélisation UML est fourni avec les trois éditions de Team System.

Ces différentes modélisations s'intègrent dans le concept « Dynamic System Initiative ». Chaque modèle, tout en restant autonome, peut interagir avec d'autres. Ainsi, les modèles, applicatif et système, vérifient leur cohérence et pertinence par rapport au modèle de déploiement. De plus, si on conçoit la logique data center, on peut, en plus, vérifier la cohérence de déploiement selon la modélisation du data center. Et bien entendu, le code généré s'adapte aux modifications apportées aux modèles. Pour la modélisation, Team System utilise le schéma System Definition Model (ou SDM), format XML contenant la description du modèle. SDM est commun aux différents modèles Team System.

SDM est un méta-model permettant de créer des modèles divers qui capturent les éléments constituant un système distribué. SDM permet une modélisation multi couche incluant l'application, le réseau (sa topographie), les périphériques et le système d'exploitation. Comme dit plus haut, avec SDM, on peut définir les contraintes, les règles de déploiement de chaque couche composant le modèle de déploiement.

Basiquement, cette modélisation passe par :

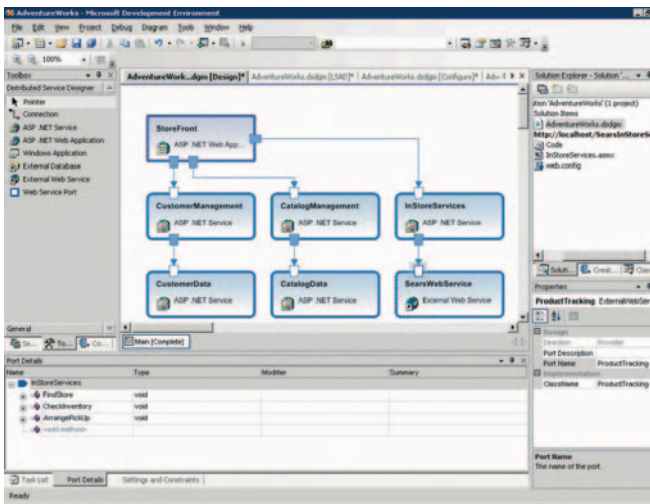
- un langage commun pour décrire le design et la configuration
- définition des requis de l'environnement runtime
- intégration dans le projet .Net
- synchronisation entre le modèle et le code
- modèle extensible.

Concepteur d'applications distribuées

Aide un développeur ou un architecte à définir et à configurer son application sur les systèmes où le projet sera déployé. Il dispose de prototypes prêts à l'emploi, notamment pour les applications web et les architectures orientées services de type SOA et web services. Il prend aussi en compte l'environnement et les composants Biztalk et les applications desktop Windows.

Il est possible de créer un modèle de zéro ou de faire du reverse engineering (à partir d'une solution ou d'un projet).

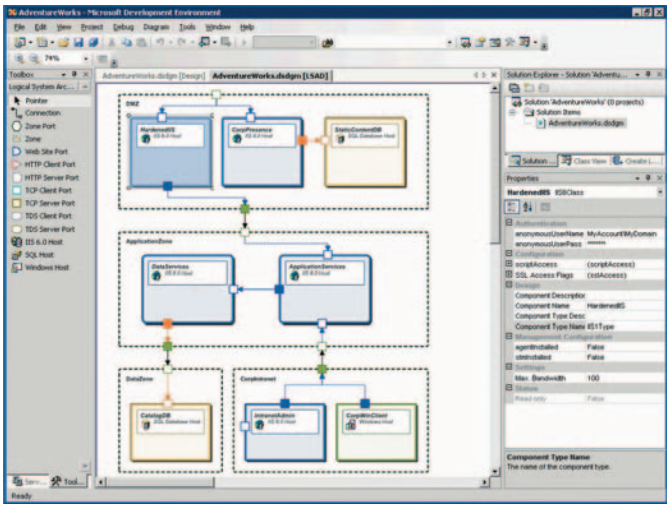
Il possède deux types d'applications : les applications dites internes et applications dites externes. Les internes sont des applications que l'on déploie individuellement. Les ressources peuvent être incluses, ou mises en référence ou partagées. Les applications externes fournissent une visualisation de références des autres systèmes qu'il faut résoudre durant le déploiement.



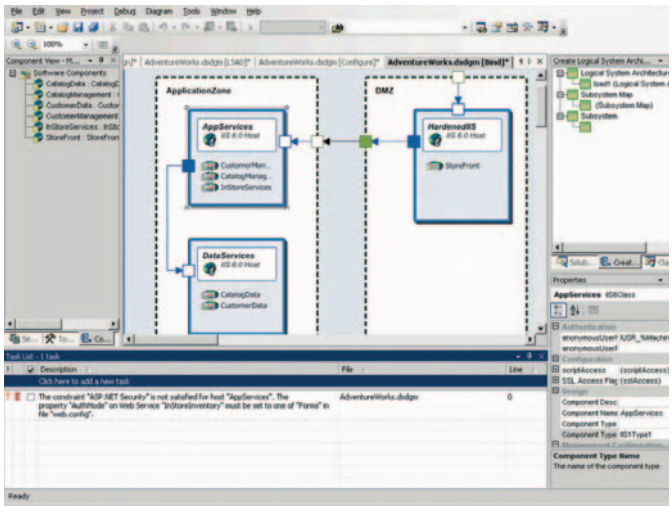
Le concepteur d'application

Le designer du logical datacenter (Concepteur d'infrastructures)

Il sert à représenter et à connecter la structure logique d'un datacenter, ou tout du moins à les documenter, car on ne fait pas de description physique des serveurs. Il fournit donc d'importantes informations sur les environnements de déploiement cibles. Ce qui se révèle souvent utile pour le développeur. Il permet de spécifier et de configurer les types de serveurs composant le datacenter, en fournissant aussi les types de communications possibles, les chemins d'accès et les services disponibles. Il est bien entendu totalement intégré à Visual Studio. Le diagramme du datacenter se crée indépendamment du processus de l'application. Ce diagramme peut se réutiliser pour d'autres systèmes distribués. Pour faciliter la création, on dispose de prototypes prêts à l'emploi. Il devient alors possible de déployer sur des sections, des serveurs qui sont réellement optimisés pour l'application à déployer, et cela peut alors procurer les performances optimales. Vous pouvez aussi mêler cela à une



Le concepteur de classe



le concepteur de déploiement

politique de sécurité non moins optimale, afin de réduire la surface d'attaque potentielle.

Pour la création de votre diagramme, dans tous les concepteurs graphiques vous disposez d'une toolbox contenant les composants pré construits : dataserver, generic server et web server. Bien entendu, l'outil est extensible. Quand on crée un modèle, par défaut, il dispose d'un serveur web pour la présentation, un pour les services et un autre pour la base de données de type SQL Server.
À partir de ce diagramme, vous pourrez effectuer les connexions nécessaires entre les différents serveurs. Pour tester le diagramme, passez par le Deployment Designer (Concepteur de Déploiement).

Pour les définir et affiner les règles, vous pourrez passer le Settings and Constraints Editor. Par exemple, vous accéderez aux options de sécurité d'ASP.Net.

Quand vous souhaitez œuvrer dans un environnent complexe, vous pourrez utiliser l'option ZONE. Une zone est un conteneur pour serveurs logiques et les autres éléments réseaux et permet d'en restreindre l'arrière plan et les paramètres de communication. On peut ainsi prendre en compte les firewall et les configurations. Il est même possible d'importer les paramètres d'un serveur IIS.

Le designer d'application (Concepteur d'applications)

Cette partie permet de décrire, grâce à un diagramme, mon application. Il comprend les applications Windows, Web, Office, services Web, etc. On peut définir des contraintes (sur les exigences déploiement, par exemple). Les types d'application proposés supportent la génération et la synchronisation avec les projets associés et le code. Cette synchronisation est bi-directionnelle. Quand on modifie, l'outil modifie l'autre.

Comme son nom l'indique, ce designer s'occupe des applications, des architectures services web / SOA., des applications Web. Par défaut, on dispose de plusieurs prototypes d'applications (dans la toolbox). Chaque prototype fournit une application de base pré-configurée. Il est donc rapide de démarrer avec un squelette basique, puis ensuite, l'étendre. Notez que l'on dispose d'un type d'application générique. Vous jouez avec les points de terminaison et les connections, par exemple pour lier deux applications.

Le designer système

Il sert à composer et à configurer les systèmes des applications qui sont définis dans le concepteur d'applications. Dans notre environnement, le système représente une unité de déploiement et sert de configuration à une ou plusieurs applications et à d'autres sous-systèmes. Car, n'oubliez jamais qu'un système peut se composer d'autres systèmes, formant ainsi des scénarios complexes de déploiement. La définition de l'unité système se fait indépendamment du concepteur d'applications. Cette partie intéressera les architectes et développeurs sans oublier les responsables de déploiement et de la production.

Modéliser le déploiement et donc les systèmes sur lesquels ont déploie, permet, dès la conception du projet, de prévoir les scénarios de déploiement, les contraintes, les requis. Cela permettra de supprimer une partie des incertitudes et erreurs lors du déploiement de l'application. Enfin, cela permettra de réduire le temps nécessaire à mettre en place le déploiement et de pouvoir l'adapter rapidement suivant l'évolution des demandes, des contraintes. Cela fait partie de Dynamic System Initiative. On passe par le System Definition Model ou SDM, meta modèle qui décrit les connexions, les configuration et les relations entre les applications et ses services mais aussi l'environnement runtime, ce qui est important notamment quand on utilise des langages s'appuyant sur des machines virtuelles (Java, .Net, etc.).

Le designer de déploiement

Il sert à définir le déploiement spécifique d'un système dans la cible d'un data center. L'application est alors « bound » du serveur logique à la cible data center. Une phase validation est requise pour confirmer le bon déploiement, en vérifier le respect des règles et contraintes du modèle. Il s'assure aussi que la communication fonctionne bien entre l'application et les serveurs. Ce modèle sert à garantir que votre type d'application est en phase avec le type de serveur logique.

Un point important : Visio-UML est inclus dans toutes les éditions Team System.

F. Tonic

Team System Foundation Server : les fondations d'un serveur durable

La première mouture de Team System Foundation bouleverse considérablement l'approche Microsoft du développement qui se cantonnait, jusqu'à maintenant, au poste de travail, en proposant un serveur clé en main, capable d'adresser la gestion des travaux, de la configuration ou de l'intégration. Cet article donne un premier aperçu des capacités de ce nouveau produit, tout en apportant les clés d'une mise en œuvre réussie.

L'arrivée, ce mois d'avril, de Team System Foundation achèvera le lancement de la gamme Visual Studio Team System. Moins spectaculaire que l'outil de développement Visual Studio 2005, regorgeant de designers et autres diagrammes d'abstraction du code source, ce serveur révolutionnera, malgré tout, notre approche du développement sur la plate-forme .Net.

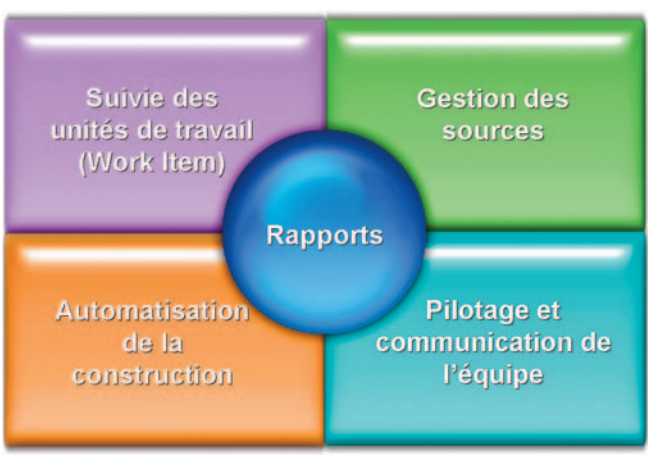
Le meilleur de l'Open Source

Avec une approche pragmatique, dans laquelle Microsoft est maintenant passé maître, l'éditeur de Redmond a synthétisé les outils d'aide au développement les plus couramment utilisés sur sa plate-forme .Net 1.1, pour en faire un premier ensemble cohérent et homogène, appelé à s'enrichir dans les semestres à venir. Fonctionnellement, les adeptes de Cruise Control, Nant, couplés à Nunit et Fxcop pourront, à première vue, ne pas trouver de raison à se précipiter vers ce nouveau serveur, qui sur certains aspects pourra paraître moins avancé que son composant open source alter ego. Cependant, au delà de tout esprit polémique, Microsoft nous livre là une plate-forme clé en main, capable d'être installée et opérationnelle dans la journée, là où un spécialiste demandait plusieurs jours à stabiliser, paramétrer et enseigner la manipulation d'outils hétérogènes, plus ou moins adaptés aux différents acteurs gravitant autour de nos projets.

Fonctionnellement, Team System Foundation adresse quatre domaines : la gestion des tâches du projet, la gestion de configuration, l'automatisation de la construction et le pilotage du projet.

Les travaux au cœur du System

Pour planifier, tracer et coordonner les tâches d'un projet, Team System Foundation introduit la notion de Work Item, comme unité de travail primordiale. Sur le modèle d'un bug tracker, les différents intervenants du projet disposent de fiches de saisie et d'un moteur de Workflow, assurant la maturation de ces fiches au sein du référentiel. Toutefois, du simple bug tracker, Team System Foundation ne reprend que le modèle, car nous disposons bien là d'un outil extrêmement abouti. Si la méthode MSF Agile prédéfinit déjà 5 types de Work Item (Bug, Tâche, Risque, Contrainte de Qualité de Service et Scénario), il est à la portée de tout administrateur du serveur de modifier ces fiches existantes ou d'en créer de nouvelles. Comme un projet informatique ne regroupe pas que des aficionados de Visual Studio, Microsoft a fait en sorte qu'Excel puisse être utilisé pour la saisie en masse de Work Items, ou Project, pour une planification affinée. Cependant, la véritable valeur de ces Work Items ne se joue pas là, mais dans le fait qu'ils peuvent être associés à chaque action sur le code source (checkin, découverte d'un problème d'intégration, correction d'un bug, ...) et toucher du doigt le doux rêve de la traçabilité des processus de développement, véritable casse-tête de nos patchworks d'outils open source.

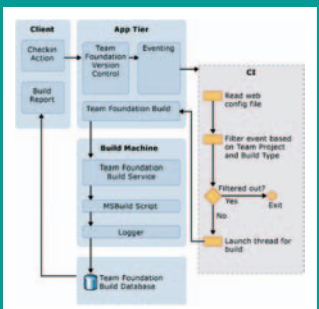


Visual SourceSafe

Côté gestion de configuration, Microsoft nous propose enfin une alternative sérieuse à Visual SourceSafe (qui existe toujours dans sa version 2005 pour les petits projets). Le code source et tout autre élément des projets Team System (diagrammes, fichiers de configuration, ...) sont maintenant sauvegardés dans SQL Server 2005, avec un accès distant par Web Services. Vous pourrez maintenant gérer les configurations de projets complexes, avec des équipes fonctionnant en parallèle, assurer le support de versions d'une application en production, tout en travaillant sur de nouvelles fonctions. De fait, Team System permet de créer et fusionner des branches de développement, en bénéficiant d'un outil graphique de résolution des conflits. Les différentes briques de la plate-forme étant liées, chaque branche de développement pourra bénéficier de ses propres logiques d'intégration, de reporting et, bien entendu, de planification.

Team System et l'intégration continue

Déceler et remonter au plus tôt la présence d'un bug constitue un enjeu majeur de productivité.



Si Team System Foundation ne fournit pas en tant que tel, de fonction d'intégration continue prête à l'emploi, il propose en revanche l'outil Team System Build, construit au-dessus de MSBuild, utilisable pour monter une plate-forme d'intégration continue, moyennant l'utilisation d'un composant additionnel tel que CI REF CI \h [1]. Dans cette configuration, un service web s'enregistre auprès du gestionnaire de contrôle de code source, pour être notifié lors de chaque checkin. Le message véhiculé contient les informations utiles au web services pour déterminer s'il y a lieu de lancer un type d'intégration, compte tenu de paramètres positionnés dans son fichier de configuration. Team System Build permet à ce moment de lancer un scénario d'intégration allant du contexte de compilation jusqu'aux configurations de tests à jouer, en passant par la vérification du respect des bonnes pratiques de développement.

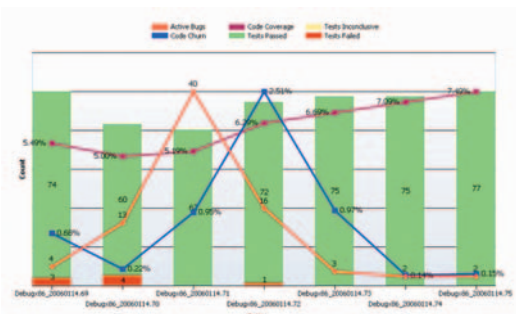
Il faut cependant bien avoir à l'esprit que la décision de mettre sur pied une plate-forme d'intégration continue ne se prend pas à la légère. Lier le lancement d'une intégration à la réalisation d'un checkin, implique de ne pas utiliser cette fonction comme une simple sauvegarde côté serveur de son code source (Microsoft a introduit, à ce niveau, la fonction de Shelving), eu égard aux pratiques observables de nos jours auprès d'un nombre, encore trop grand, de développeurs. De plus, pour des équipes avoisinant la trentaine de développeurs, la fréquence des checkin pourra nécessiter l'utilisation d'un serveur Team System Build dédié, pour maintenir un niveau de disponibilité satisfaisant du serveur Team System Foundation.

Intégration et automatisation

Team System Build est un outil additionnel de la plate-forme, utilisé pour l'automatisation de la phase d'intégration. Dans une approche d'industrialisation des développements, où intégration rime avec compilation, test unitaires, tests de montée en charge ou respect des bonnes pratiques de développement (voir encadré Team System et l'intégration continue), cette brique occupera une place prépondérante, justifiant son installation sur un serveur dédié.

De la méthode

Disposer d'outils intuitifs et intégrés à l'environnement de développement réhabilite la méthode. Les premières expériences de projets réalisés avec Team System montrent que les contraintes de fonctionnement de l'outil passent rapidement au second plan, l'attention se portant (pour la première fois dans certaines équipes) sur l'amélioration de la capitalisation dans les phases amont au développement (voir encadré Team System et la prise en compte des exigences) ou le reporting du projet. Placer SQL Server 2005, et en l'occurrence Reporting Services, au cœur de Team System Foundation dote le chef de projet d'un tableau de bord entièrement modulable, pour l'accompagner dans sa fonction de pilotage.



Team System Foundation pour tous

Parce qu'il y a autant d'organisations autour d'un projet que d'équipes projets, si ce n'est d'acteurs sur ces projets, Microsoft a laissé ouvert l'accès à Team System Foundation, via une batterie de Web Services. Chaque module du serveur expose son jeu de Web Services, permettant de réécrire tout ou partie des fonctions natives de Visual Studio Team System. Certains éditeurs (www.devbiz.com, www.teamprise.com...) proposent déjà un certain nombre de plug-ins pour Eclipse, Visual Studio 6.0 ou .Net 2003, afin de bénéficier des services offerts par Team System Foundation depuis un environnement Java ou Visual Basic. Cette approche est d'autant plus pertinente, dans nos entreprises rarement mono-technologiques, que Team System apporte pour la première fois la possibilité de fédérer les méthodes de développements entre plates-formes.

Team System Foundation mis en œuvre

Si partir sur Team System ne représente pas une gageure du point de vue technique, cela demande tout de même le recours à un projet pilote ou preuve de faisabilité (POC), moment clé pour déterminer la capacité des équipes à évoluer vers l'industrialisation logicielle. Les pratiques en vigueur dans l'entreprise et les niveaux d'expériences de chacun doivent en effet déterminer le degré d'implication de Team System dans les processus de développement de l'entreprise, plus que tout élan volontariste dévastateur.

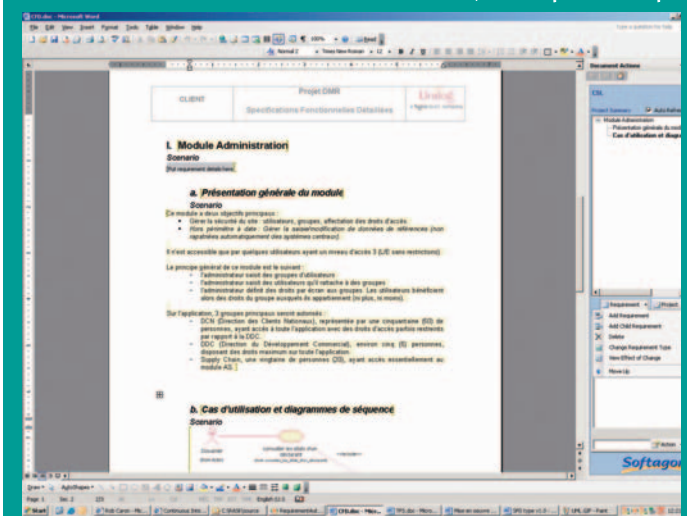
Un objectif de POC identifié, il vous faudra passer par l'étape d'installation de la plate-forme, avant le choix des outils et la résolution du POC proprement dit.

L'installation de la plate-forme REF MORPHEUS \h [3] nécessite l'utilisation d'un serveur dédié,

Team System et la prise en compte des exigences

Comment proposer Team System à des responsables utilisateurs ou à une MOA.

Team System Foundation revendique une certaine ouverture sur l'extérieur de la sphère habituelle de l'équipe projet, pour se tourner vers des publics plus fonctionnels, en témoigne l'utilisation de Project ou Excel, pour accéder au module de gestion des Work Items. Cependant, si la gestion des exigences fonctionnelles, adressée par des outils tels que Caliber RM de Borland, est la problématique au cœur des préoccupations d'une MOA, vous ne trouverez pas d'éléments probants à ce sujet dans Team System Foundation, présenté dans son plus simple appareil. Pour pallier à ce manque, Softagon vous propose un exemple d'application réalisable autour de Microsoft Word REF SOFTAGON \h [2].



En partant du constat qu'un responsable utilisateur souhaitera utiliser un éditeur de texte, plus que tout client léger élaboré, pour décrire son besoin (moyennant plusieurs affinages), Softagon a développé un modèle de document via VSTO, capable de gérer la hiérarchie du document et, le moment échéant, de la sauvegarder en base de données. Dans l'optique de tirer parti du meilleur de chaque monde, SharePoint nous permettra d'historiser le document avant sa synchronisation avec Team System Foundation. Cette dernière étape, typiquement réalisée suite à la validation finale du document, automatise la création de Work Items purs et durs, associés un à un aux paragraphes du document. Chacun de ces Works Items réputés fonctionnels pourra ensuite être associé à une série de Work Items plus techniques, pour être pleinement utilisables par les équipes opérationnelles.

Pour aller plus loin, nous pourrions proposer un suivi de la prise en compte des exigences. Là encore, en intégrant SQL Report Server, Team System Foundation ouvre la voie au développement d'un composant de reporting qui permettra, par exemple, d'évaluer le pourcentage d'avancement d'une exigence fonctionnelle en agrégeant les avancements individuels des tâches de réalisations associées.

Team System Foundation demandant au bas mot 1 Go de mémoire vive et un processeur dernière génération. Si les versions bêta de Team System Foundation imposaient une installation sur un contrôleur de domaine Active Directory, ce qui pouvait poser des problèmes délicats au moment d'impliquer les utilisateurs dans un POC, la version actuelle supporte à la fois les configurations Active Directory (simple membre d'un domaine) et Workgroup. Suivront ensuite, les installations de Windows SharePoint Services 2003 (et ses services pack), SQL Server 2005 et finalement Team System Foundation. Notons que le media d'installation de Team System Foundation contient les installations du serveur, du service d'intégration (Team System Build) et du client (Team Explorer), à déployer sur l'ensemble des postes utilisateur, pour accéder au serveur depuis Visual Studio Team System, Excel, Project ou toute autre application basée sur les Web Services de Team System Foundation.

Conclusion

Cet article a tenté de dresser un panorama des domaines adressés par Team System Foundation, pour donner la mesure de la tâche à laquelle s'est attaquée Microsoft. Sans révolutionner pour le moment ces domaines, individuellement parlant, ce produit apporte néanmoins une profonde structuration du développement sur la plate-forme .NET, en posant les fondations de nos usines de développement, accompagnées des lignes directrices pour mener à bien la construction du reste de l'édifice.

Ressources

[1] http://msdn.microsoft.com/library/en-us/dnvs05/html/ConIntTmFndBld.asp
[2] http://directory.partners.extranet.microsoft.com/SolutionPage.aspx?i=111&SolutionID=4484
[3] http://morpheus.developpez.com/vsts/
[4] http://blogs.msdn.com/robcaron/

Arnaud FONTAINE
Unilog IT Services (LogicaCMG company)

L'identification d'un POC

Quel meilleur moyen pour convaincre de la pertinence de Team System que de démontrer la faisabilité d'un problème tenace dans l'entreprise !

Les coûts d'acquisition d'une plate-forme telle que Team System, et les bouleversements organisationnels qu'elle peut induire, font que le choix de son adoption dépasse le plus souvent les équipes informatiques. Aussi la démarche la plus constructive consiste à prouver la valeur ajoutée de Team System sur les problématiques récurrentes de nos projets (planification, anticipation, valorisation de l'information...), en ayant recours à un démonstrateur mis en place en quelques jours.

Votre potentiel, notre passion.™

Microsoft®

McCann

Une compagnie de fret maritime qui gère 15 milliards de transactions par an.
Avec Microsoft SQL Server 2005.

Mediterranean Shipping Company, la 2e entreprise de porte-conteneurs au monde, affrète 7 millions de conteneurs depuis 116 pays. Pour sa base de données stratégique de 5 térabytes, elle a choisi le nouveau SQL Server™ 2005, qui fonctionne sous Windows Server™ 2003 et qui a une disponibilité de 99,999 %*.
Découvrez comment sur **www.microsoft.com/france/sql**

Microsoft® Windows Server System™

* Résultat variable. La disponibilité dépend de nombreux facteurs, comme le matériel, les logiciels, les processus opérationnels critiques et les services aux entreprises.