MSDN Webcast:

TFS 2008: Mit definierten Buildprozessen und Continuous Integration zu mehr Softwarequalität

Visual Studio Team System (Teil 5 von 10)

Veröffentlicht: 19.03.2008

Presenter:
Neno Loje, MVP für Team System
www.teamsystempro.de

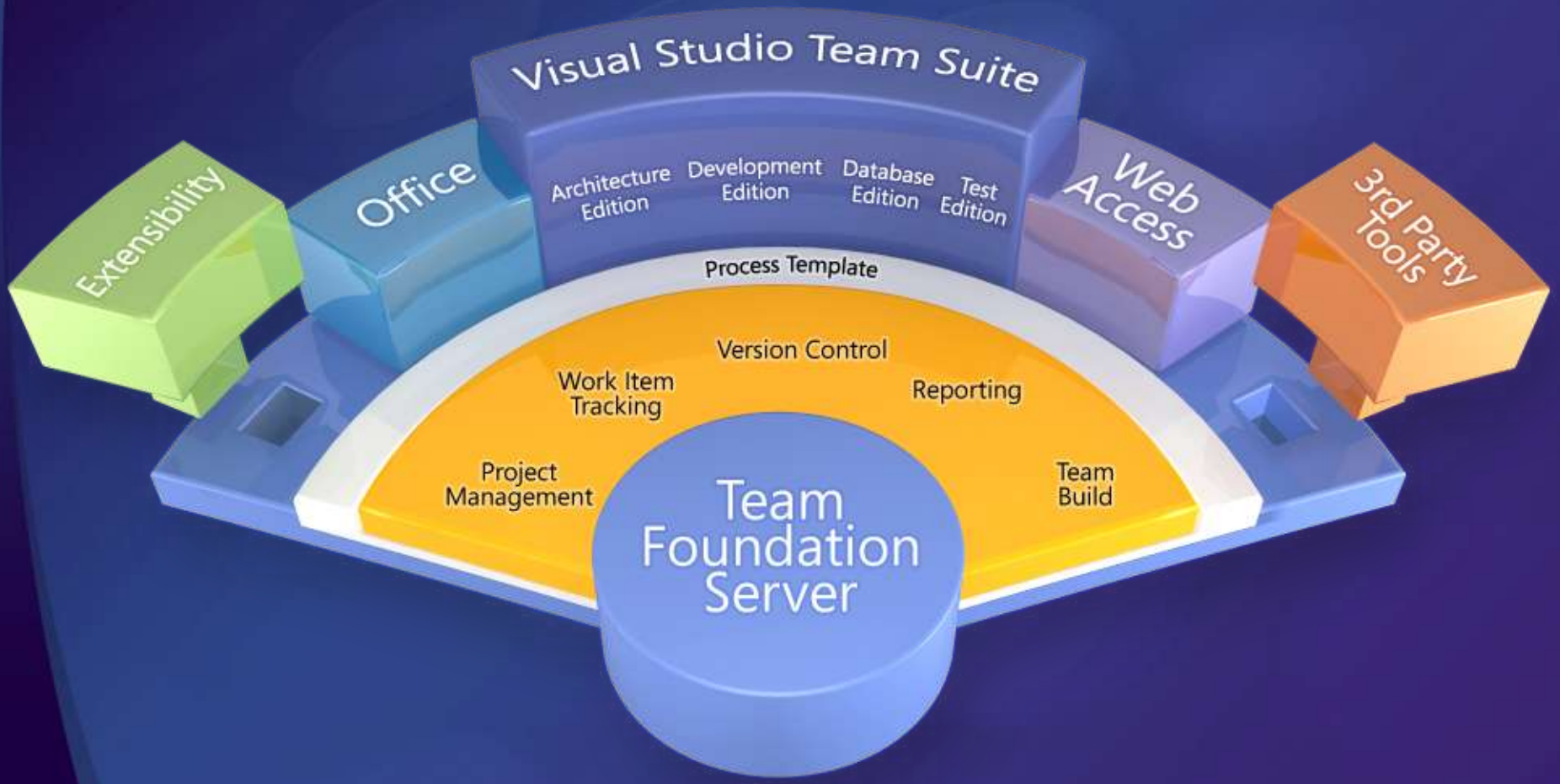
Voraussetzungen

- Sie sind jemand, der Code baut!
  - Software-Entwickler, „Release Manager“, …
- Sie wollen vielleicht…
  - … den Vorgang des Codebauens automatisieren?
  - … erfolgreich hohe Qualität sicherstellen durch Tests als Teil des Codebauens?
  - … Werkzeuge kennenlernen, die Ihnen das Bauen leichter machen?

Agenda

- Begriffe
  - Buildprozesse, -skripte und Builds
- Motivation
- Typischer Buildprozess
- Werkzeuge für Buildprozesse

Visual Studio Team System



Begrifflichkeiten

- Integration
  - Alle Änderungen in einen Topf werfen
- Buildprozess
  - "Build Definition"
- Buildskript
  - "Build Type" oder "Build Project" (.proj)
  - MSBuild
- Build
  - interner vs. freigegebener Build

Was interessiert mich das?

- Typische Integrationsphase in einem Projekt
  - Lang
  - Hart
  - Anstrengend
  - Sie möchten so schnell wie möglich fertig sein
  - Sie fühlen sich mies
- Sie würden gerne…
  - … weniger Arbeiten
  - … mehr Geld verdienen
  - … sich gut fühlen!

Was ist damit gemeint?

- Integrieren Sie Ihren Code
  - Automatisierter Build
  - Tests laufen lassen
  - An diverse Orte verteilen (Test, Staging)
  - Review der bei Intergration aufgetauchten Fehler
- Machen Sie es regelmäßig!
  - Bei jedem Check-In
  - Alle *X* Minuten

Buildautomation

- Erfordert keine Interaktion von menschlichen Wesen
- Try
  - Aktuellen Stand aus Sourcecodeversion laden
  - Alle Konfigurationen (von Grund auf) kompilieren
  - Qualitätsprüfungsprogramme ausführen
    - Unit Tests, Codeanalyse, Integrationstests
  - Verteilen in Test / Staging-Umgebung
  - Erfolg berichten
- Catch (Exception)
  - Misserfolg berichten
    - Kompilierung fehlgeschlagen
    - Tests fehlgeschlagen

Buildautomation: Werkzeuge

- Visual Studio Team System
  - MSBuild
  - Team Foundation Server – Build Server
- Kommerzielle Drittanbieterwerkzeuge
  - FinalBuilder
  - Visual Build
  - …
- Freie/Open-Source Werkzeuge
  - NAnt
  - Cruise Control .NET
  - …

Warum bauen wir Code?

- Was meine ich mit „bauen“?
  - Manchmal ist es wirklich nur das Kompilieren…
  - In vielen Fällen aber wesentlich komplexer
- Ganz einfach:
  - Wir wollen das Programm ausführen um es zu testen
  - Wir wollen das Programm beim Kunden laufen lassen
- Weitere Gründe?
  - Wir wollen ein hohes Maß an Qualität sicherstellen
    - Wir wollen bei neuen Version oder Updates ruhig Schlafen können
- Wann bauen wir Code? Die ganze Zeit!

Häufige Missverständnisse

- "Einen Buildprozess braucht man erst ganz am Schluss"
- "F5 ist genug"
- "Bauen == Kompilieren"

Was ist ein Build?

- Eine mögliche Definition:
  - Ein auslieferbarer Stand des Produkts (Programmdateien, Setup, Dokumentation, dazugehörige Tests, etc.)
- Verwandte Begriffe: *Release* (= ausgelieferter Build)
- Ein Build wird durch einen *Buildprozess* erzeugt
- Ein *Buildprozess* ist die Summe aller Aktionen/Befehle eines Skripts um das Ergebnis (= Build) zu erreichen
  - Die Aktionen/Befehle werden in einem Buildskript definiert
  - Das Buildskript kann z.B. in einem Batchfile definiert sein

Wie viele Builds gibt es?

- Es kann mehr als einen Build geben
  - Das ist auch der Normalfall
  - Beispiel für ein Produkt:
    - Daily Build – regelmäßiger Buildprozess
    - Internal Build – für den internen Gebrauch
    - External Build – für die Auslieferung zum Kunden

Was sind typische Bestandteile?

1. Initialisierung & Quelltexte aus Versionsverwaltung laden
   - Neue Versionsnummer ermitteln, aktuelle Quelltexte laden
2. Programm und Dokumentation bauen
   - Kompilierung, Hilfedateien, Dokumentation, Installationspaket
3. Qualitätskontrolle
   - Genügt dieser Build unseren Qualitätsansprüchen?
   - Unit Tests, Statische Codeanalyse, Dynamische & Lasttests
4. Auslieferung & interne Archivierung
   - Kompression (z.B. ZIP)
   - Kopieren in Testumgebung oder Produktivsystem
   - Neue Version ins Internet stellen (z.B. per FTP)
   - CD/DVD-Image vorbereiten (z.B. ISO)
5. Abschluss
   - Neuen Build in eine Liste eintragen (z.B. Datenbank)
   - Benachrichtigungen (z.B. E-Mail, Messenger, ...) abschicken

Kompilieren klingt so einfach…

- Abhängigkeiten zwischen Solutions
  - Nicht von Visual Studio erfasst…
- Reihenfolge beachten
  - Bei mehreren VS-Solutions mit Abhängigkeiten: kleine Zyklen mit
    1. Kompilieren
    2. Testen
    3. Kopieren an Ort, auf den nächste Solution referenziert
- Gemeinsam genutzte Komponenten
  - Gehören Komponenten (DLLs, …) in die Versionsverwaltung?
  - Sollen neue Versionen von Komponenten sofort in alle Applikationen integriert werden? (auch bekannt als *Sharing*)
  - Sollen neue Versionen erst durch den Entwickler freigegeben werden müssen, nachdem er die Kompatibilität getestet hat?

Da wollen wir hin...

- Häufige Builds
  - als Integrationstest
  - als Feedback und Dokumentation des Fortschritts an den Kunden
- Automatisierte Builds
  - Wiederholbar
  - Keine Abhängigkeiten

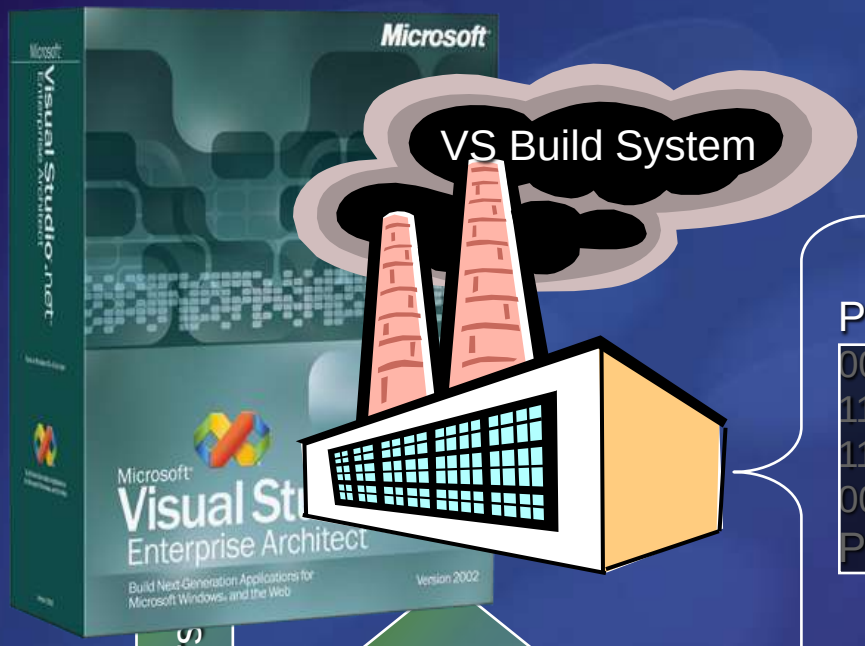
Ein neuer Build ist da!

- Wen interessiert's?
  - Den Tester: *kann endlich loslegen*
    - Aber: Was ist eigentlich neu/geändert?
  - Den Kunden
    - Aber nur, wenn der Build eine gewisse Qualität hat.
  - Den Entwickler
    - Welche Qualität hat der Build? Erreichen wir die Vorgaben?
  - … und wer auch immer sich für einen neuen Build interessiert (Benachrichtigung)
    - z.B. Ersteller eines Bug Requests möchte wissen, dass ein Build verfügbar ist, in dem der Fehler behoben wurde.

Demo

Eigener Buildprozess als Batchdatei

Visual Studio .NET 2002/2003



VS Build System

Pre build step

0011010101
1110010110
1101100111
0010100111

Post build step

Produces Abracadabra

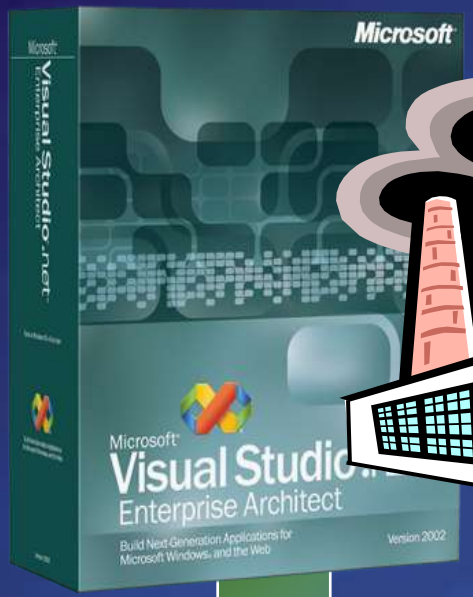
Authors

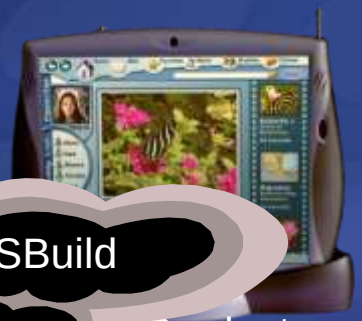
Feeds

PROJECT FILE
- $%#^$&%
- @$#%$^#

Final Product

Visual Studio 2005 und MSBuild





MSBuild

al Product

Produces

Authors

Authors

```
PROJECT FILE
<Project>
    <Property … />
    <Item … />
    <Target … />
</Project>
```

Feeds

DEVELOPER

Demo

Eigener Buildprozess mit MSBuild

MSBuild in 5 Sätzen

- … ist das Buildsystem hinter Visual Studio 2005/8
- Der Buildvorgang wird durch ein offengelegtes XML-Dateiformat definiert
- Der Visual Studio 2005/8-Buildprozess ist vollständig anpassbar
- Der Buildprozess wird durch das Schreiben von eigenem .NET Code (Tasks, Loggers) erweitert
- Sie brauchen keine IDE um Visual Studio-Projekte zu erstellen

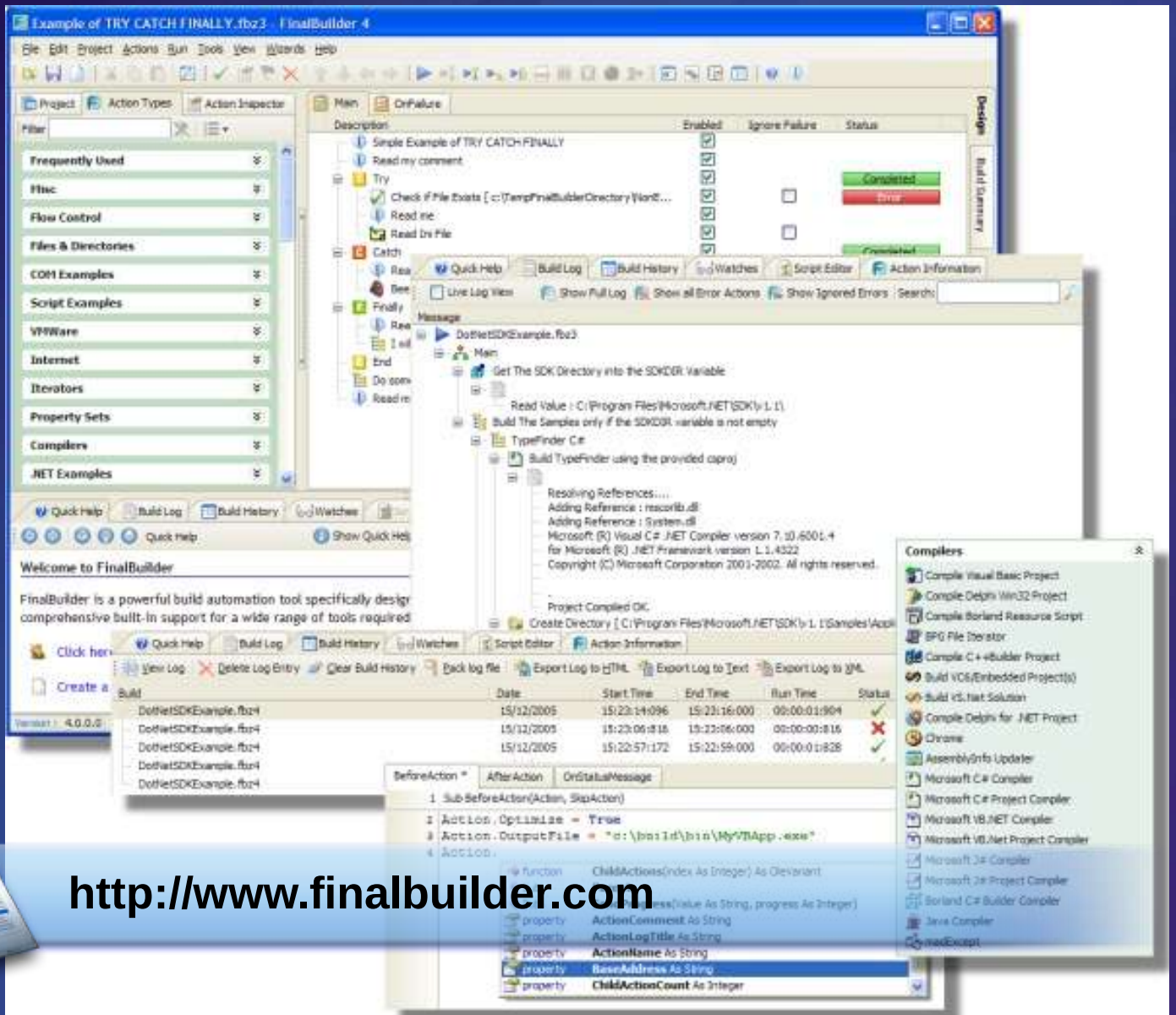
MS-Build Limitationen

- Buildskripte werden lang und unübersichtlich
  - Das Buildskript der bekannten Open Source-Komponente "NUnit" umfasst ausgedruckt 36 Seiten…
- Limitationen in MSBuild
  - Keine grafische Skripterstellung
  - Kein grafisches Debugging
  - Nichtverfügbarkeit umfangreicher Bibliotheken

Demo

Eigener Buildprozess mit FinalBuilder (€)

FinalBuilder (€)



Vorteile von FinalBuilder (vs. MSBuild)

- Grafische Skripterstellung
  - Wenige Klicks statt MB-große XML-Dateien…
- Grafisches Debugging
  - Ein Klick statt umfangreiches Aus-/Einkommentieren
  - Statusindikatoren: rot/grün für jeden Einzelschritt
- Verfügbarkeit umfangreicher Bibliotheken
  - Eingebaute Bibliothek mit 440 Aktionen
- Integration mit MSBuild und Team Build
- Preis-/Leistungsverhältnis
  - Professional Edition (1 benannte Person): 499 US$*

* Stand: März 2008

Kommt mit > 600 Aktionen:

- Actions operating on FB Variables
- PropertySet Actions
- Interactive actions
- Registry and INI actions
- Archiver actions
- Build Tool actions
- Internet actions
- .NET Tool actions
- Install Builder actions
- Virtualization Software actions
- IIS Actions
- Testing Tool actions
- Compiler actions
- Licensing actions
- Help Compiler actions
- Version Control Systems actions
- Flow control actions
- Miscellaneous actions
- Windows OS actions
- Files & Directory actions
- Iterator actions
- XML actions
- CD/DVD burner actions
- Database actions

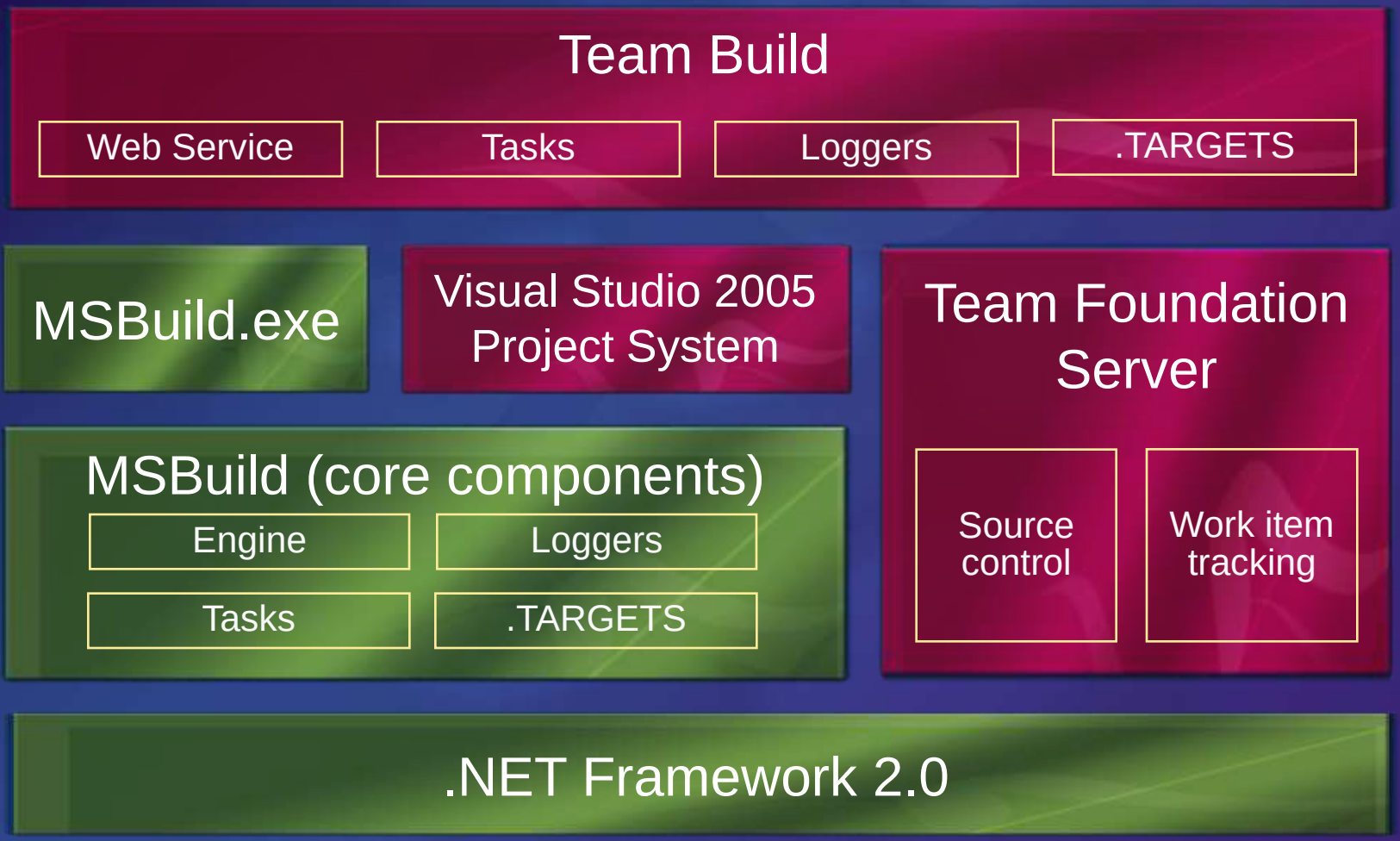
Was fehlt?

- Buildserver / Zentrale Lösung
- Integration mit Aufgabenverwaltung
- Integration in Projektverwaltung

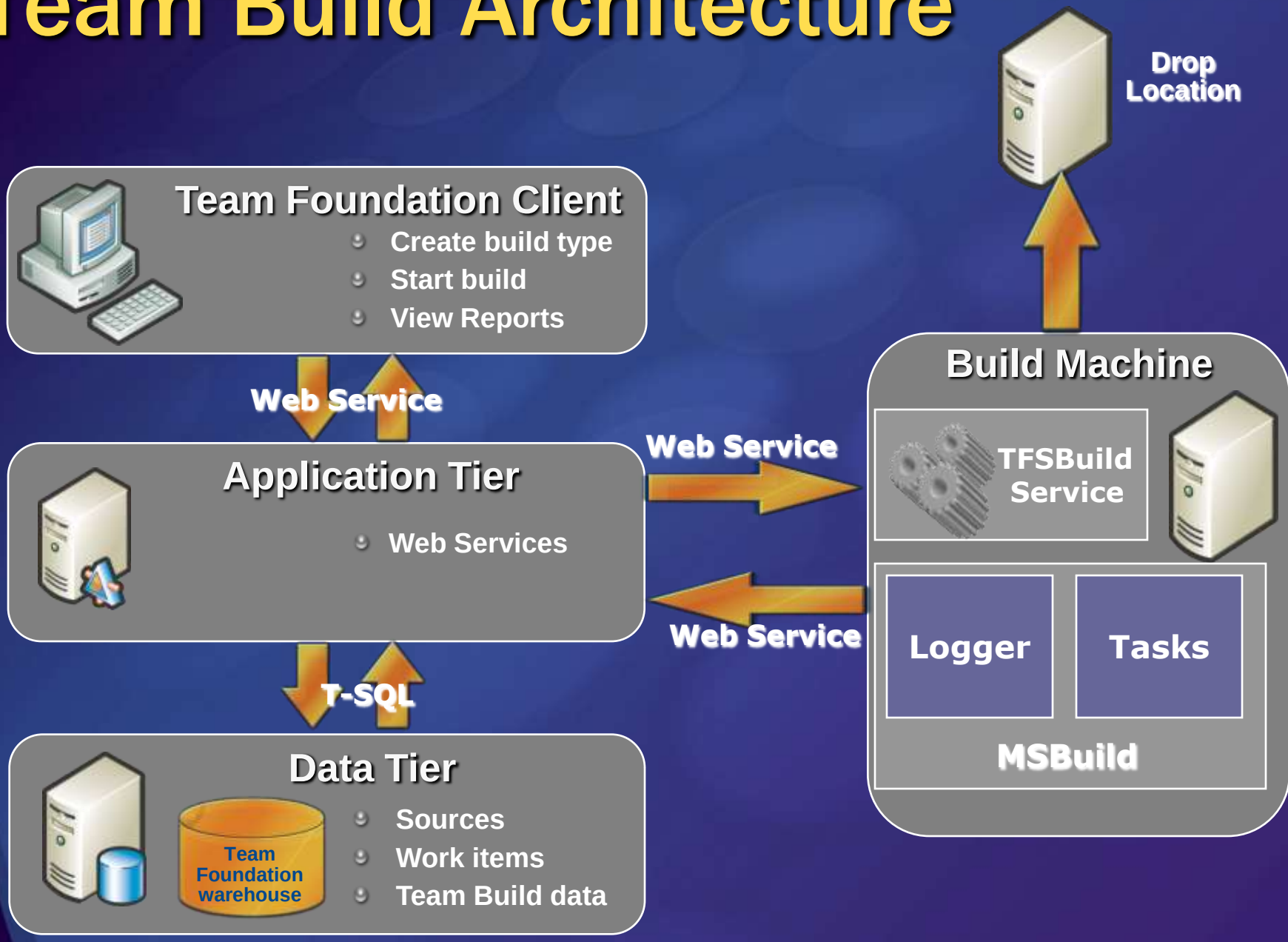
Demo

Eigener Buildprozess mit dem Team Foundation Server

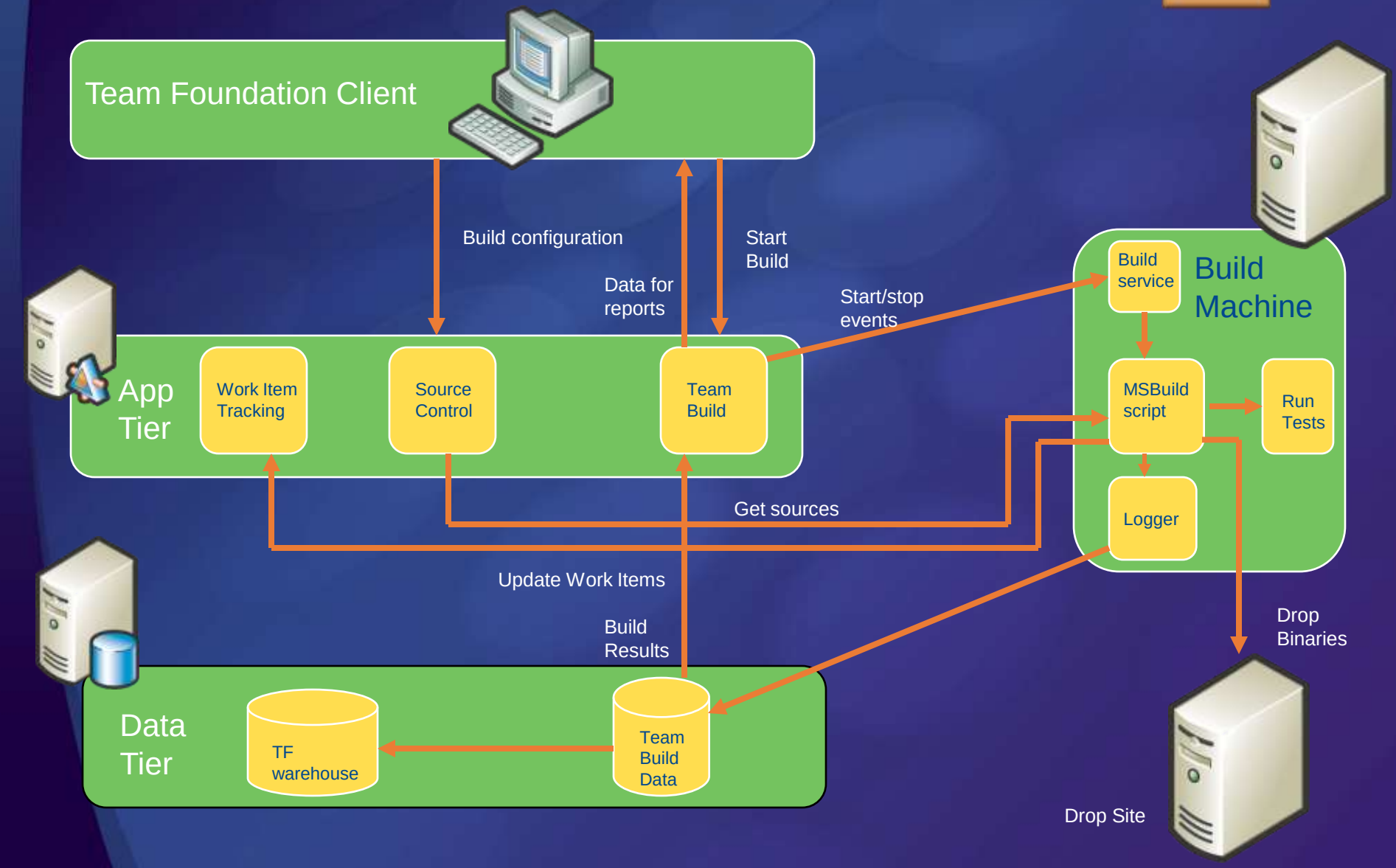
MSBuild und Team Build



Team Build Architecture



How does it work?



Buildautomatisierung

1. Build initiated from server
2. Create a build ID
3. Document build environment
4. Sync sources and tools

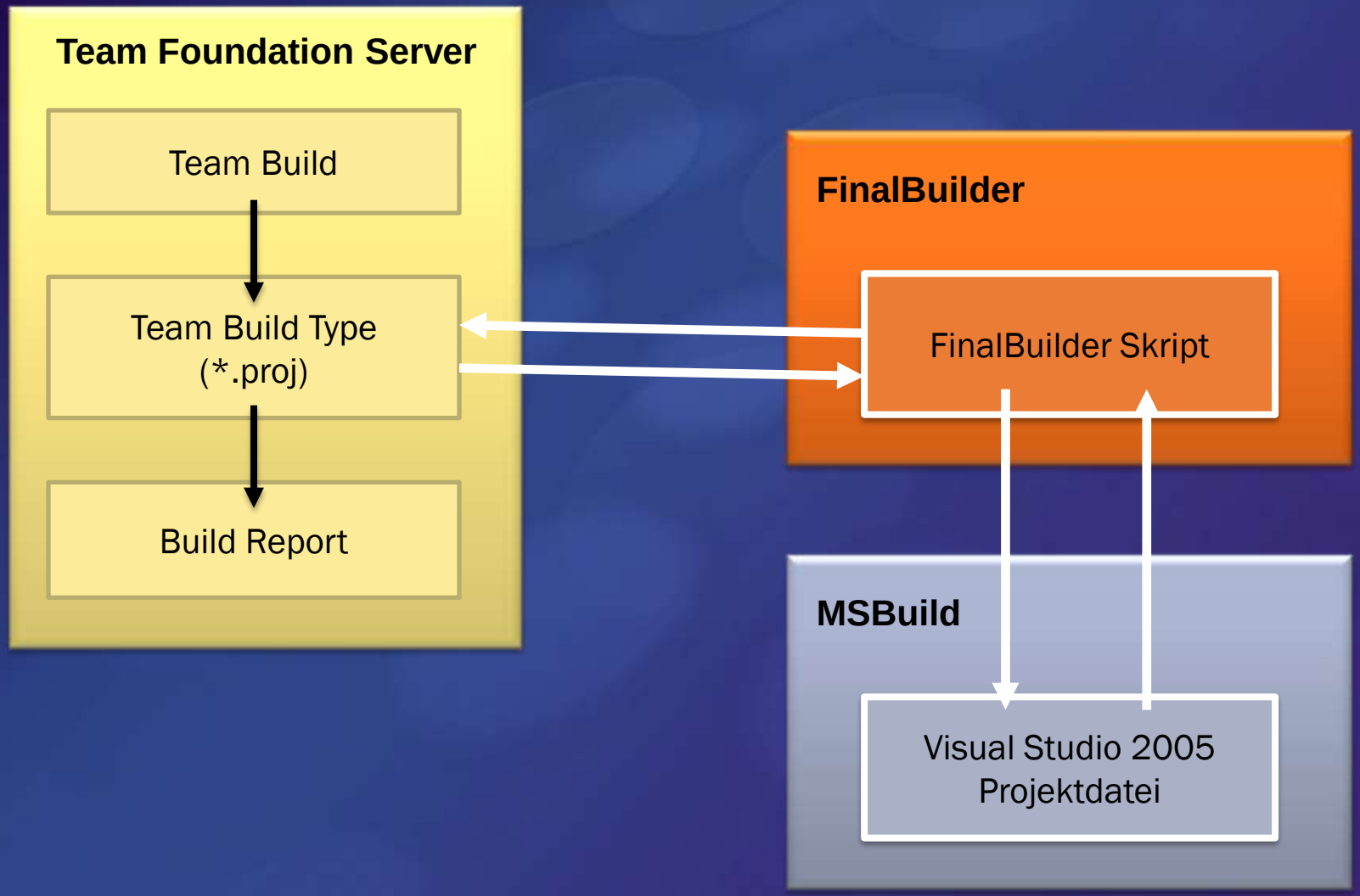
5. Compile and analyze
6. Execute tests
7. Update work items

8. Calculate code coverage
9. Calculate code churn
10. Produce build report
11. Publish build

Warum Team Foundation Server?

- Weniger Buildbrechungen
  - Beim Einchecken werden vorgegebene Regeln überprüft
  - So kommt problematischer Code seltener überhaupt in das Versionsverwaltungssystem
- Integration
  - Das Buildsystem kann mit der internen Aufgabenverwaltung kommunizieren und kümmert sich eigenhändig um
    - die Liste an Neuerungen in einem Build
    - die Aufgaben mit der korrekten Version auszustatten, in der sie implementiert / behoben wurden.
- Weil ich den TFS im Alltag nutze…

FinalBuilder und Team System



Integration mit .NET/Team System

- FinalBuilder kompiliert Visual Studio Solutions und Projekte mit MSBuild
- Aus einem Build Type im Team Build kann ein FinalBuilder-Skript aufgerufen werden
  - Die Logik befindet sich vollständig im FinalBuilder
  - MSBuild ist weiterhin für die Kompilierung zuständig
  - Team Build ermittelt weiterhin die Checkins und geänderten Work Items
- Schreiben eigener Aktionen mit .NET-Code

Was ist Continous Integration?

- Nach jedem Einchecken, sollte ein Build erzeugt werden, um dem Entwicklern **direktes Feedback** zu geben, dass seine Änderungen in Ordnung waren.
  - Denn es ist am effizientesten, wenn der, der den Build „gebrochen“ hat, diesen auch wieder gerade biegt!
- Typischer Ablauf:
  1. Entwickler „Emil“ beginnt mit der Arbeit in lokalem Workspace
  2. Währenddessen checkt ein anderer Entwickler seine Arbeit ein
  3. Emil prüft seine Änderungen lokal (Codeanalyse, Tests, ...)
  4. Nach bestandener Prüfung checkt Emil die Änderungen ein
  5. Bei CI wird jetzt ein (kompakter) Buildprozess angestoßen
  6. Bei Fehler (Kompilierung oder Tests) wird Emil sofort informiert

Vorteile von Continuous Integration

- Es fällt sofort auf, wenn der Build gebrochen wurde
  - Minimale Ausfall-/Wartezeiten
- Das Problem geht an den Verursacher zurück
  - In der Praxis muss sonst häufig ein Kollege die Fehler beheben, weil der Verursacher bereits nach Hause gefahren ist, da er von den Problemen nichts bemerkt hat.

An was man denken sollte...

- Wie groß ist mein Projekt?
- Kann ich nach jedem Einchecken einen Build anstoßen?
  - ... oder dauert das vielleicht zu lange?
  - ... kann ich den Buildprozess (nur für CI) straffen?
- Was ist, wenn kurz hintereinander eincheckt wird?
  - ... Warteschlange und hintereinander abarbeiten?
  - ... oder mehrere Buildserver verwenden?

Herausforderungen des Alltags

- Testen Sie auch wirklich alle Plattformen, die sich untersetzen?
- Haben Sie automatisierte Tests?

Unser Build-Labor

- Buildprozess
  - FinalBuilder 5.0
- Virtualisierungslösung
  - Microsoft Visual Server 2005 R2
  - Kostenlos auf www.microsoft.com zum Herunterladen
- Virtual Machines
  - Windows XP SP2 (englische Version)
  - ...

Demo

Nutzung von Virtualisierung zum Test unterschiedlicher Zielkonfigurationen

Zusammenfassung

- F5 auf dem lokalen PC reicht nicht aus
  - Lokale Konfiguration könnte Ergebnis verfälschen
  - Integrationsprobleme treten erst dann auf, wenn Sie die Arbeit aller Mitarbeiter zusammenwerfen
- Ein Buildprozess sollte nicht erst am Schluss definiert werden
  - Frühzeitige Rückmeldung um die spätere Integration zu erleichtern
- Ein Buildprozess ist mehr als nur Kompilierung
  - Klein Beginnen, kontinuierlich ausweiten

Nächste Schritte

1. Formalisieren Sie Ihren Buildprozess!
   - Minimieren Sie den "Busfaktor"
   - Integrieren Sie kontinuierlich
   - Klein anfangen, später ausbauen
2. Richten Sie Builds frühzeitig ein!
   - Diverse Auswertungen sind davon abhängig
   - Spätestens ab Tag 3 zumindest Kompilierung
3. Bauen Sie den Buildprozess sukzessive aus!
   - Mehr Schritte, mehr Zielplattformen, etc.

Weitere Informationen

- Team Development with TFS Guide – Pattern & Practices
  http://ww.codeplex.com/TFSGuide
- Operations Guidance for Team Foundation Server
  http://msdn2.microsoft.com/en-us/library/bb663036(VS.80).aspx

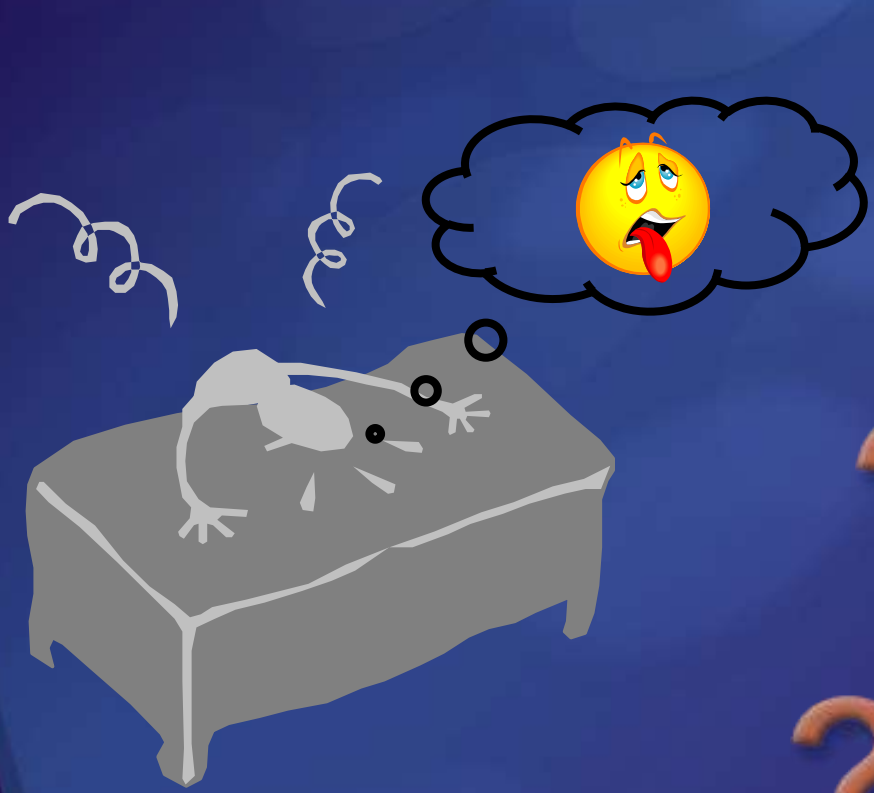
Webcast Serie zu VSTS 2008

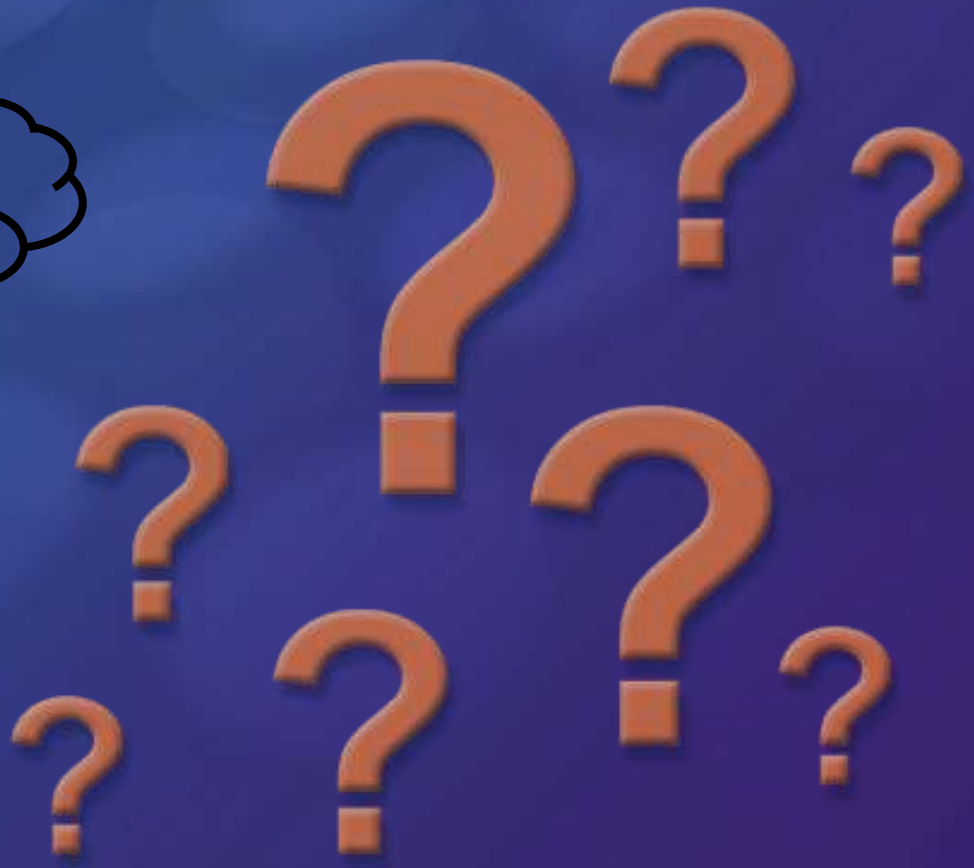
Visual Studio Team System 2008 (Teil 1 von 10) - Team Foundation Server - Mehr als nur eine Versionsverwaltung! Presenter: Neno Loje | Typ: Serien-Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	100	-	20.02.2008
Visual Studio Team System 2008 (Teil 2 von 10) - Team Foundation Server 2008 - Was ist neu? Presenter: Neno Loje | Typ: Standard Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	100	-	27.02.2008
Visual Studio Team System 2008 (Teil 3 von 10) - Auf geht's: Installation und Migration auf den Team Foundation Server 2008 Presenter: Neno Loje | Typ: Serien-Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	100	-	05.03.2008
Visual Studio Team System 2008 (Teil 4 von 10) - TFS 2008 für Entwickler: Arbeiten mit der neuen Versionsverwaltung Presenter: Neno Loje | Typ: Serien-Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	200	-	12.03.2008
Visual Studio Team System 2008 (Teil 5 von 10) - TFS 2008: Mit definierten Buildprozessen und Continuous Integration zu mehr Softwarequalität Presenter: Neno Loje | Typ: Serien-Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	200	-	19.03.2008
Visual Studio Team System 2008 (Teil 6 von 10) - TFS 2008 für Projektleiter: Projektmanagement, Arbeitsaufgaben und Berichte Presenter: Lars Roith | Typ: Serien-Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	100	-	27.03.2008
Visual Studio Team System 2008 (Teil 7 von 10) - Datenbankentwicklung als Teil des Softwareentwicklungsprozesses Presenter: Dariusz Parys | Typ: Serien-Webcast | Technologiebereich: | Dauer: N/A	Teamentwicklung, Quellcodeverwaltung und Testen	200	-	02.04.2008
Visual Studio Team System [illegible] (Teil 8 von 10) - Qualitätssicherung und [illegible]	Teamentwicklung, [illegible]	100	-	09.04.2008

microsoft.de/msdn/webcasts/serien/MSDNWCS-0802-01.mspx

Noch Fragen?:

neno@teamfoundationserver.de





Microsoft®