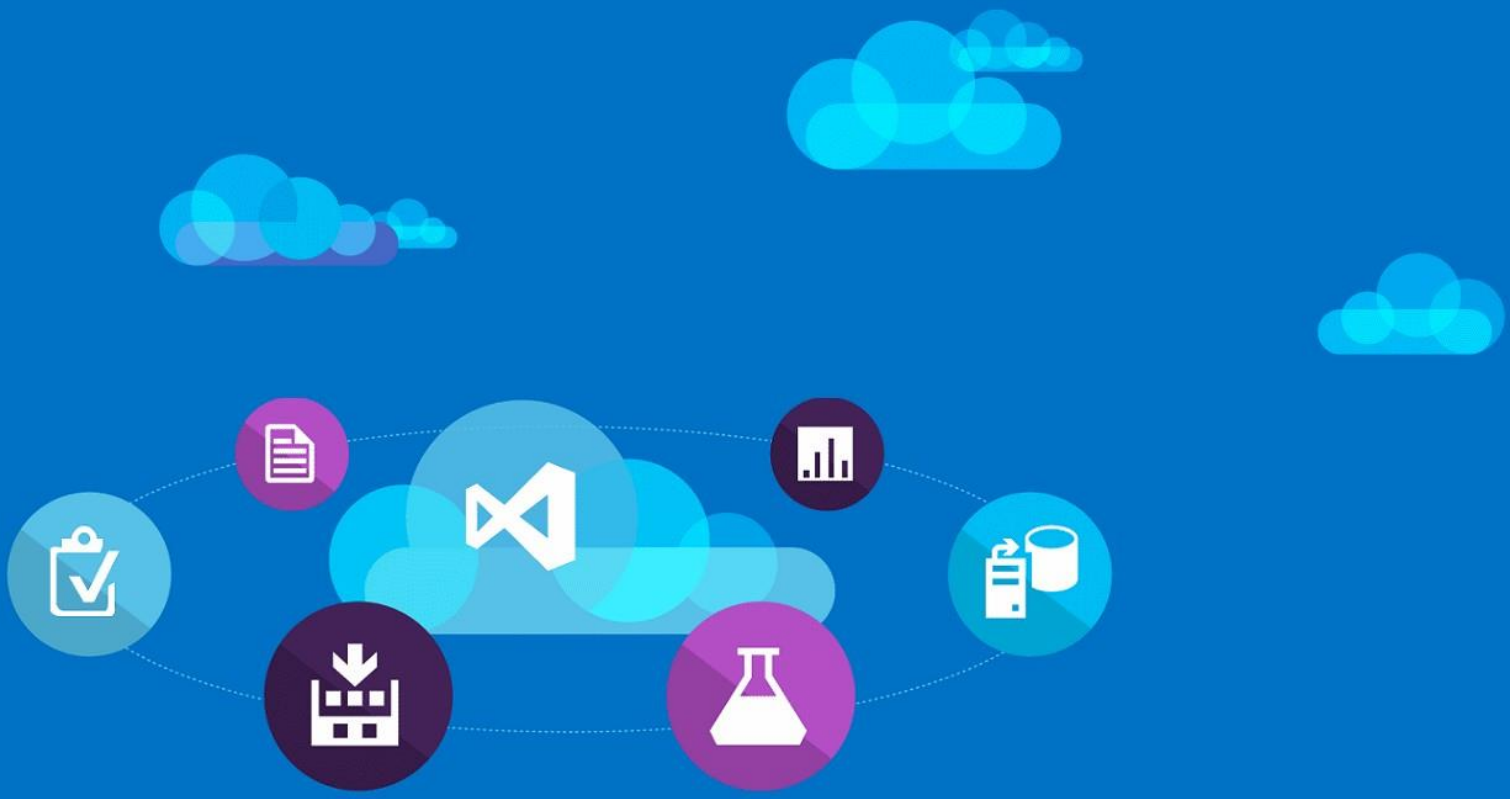


クラウドのリズムをつかむまで: マイクロソフト デベロッパー部門が学んだこと

Sam Guckenheimer

本書では、オンプレミス ソフトウェアを何年ものサイクルをかけて提供してきた「パッケージ型」ソフトウェア企業が、パブリック クラウドから継続的デリバリを行う SaaS プロバイダーとなるまでの過程で学んだことの一部をご紹介します。DevOps 手法への取り組みに焦点を当てつつ、アジャイルの次の 10 年に向けて進化させるべきツールと文化についても取り上げます。

アジャイルから DevOps へ:
マイクロソフト デベロッパー部門の進化

2013 年 11 月 13 日、マイクロソフトは Visual Studio 2013 をリリースし、併せてホステッド サービス Visual Studio Online (VSO) の購入条件を発表しました。ところがこの発表直後に、7 時間に及ぶサービス停止が発生しました。当時マイクロソフトはサービスを単一のスケール ユニットで実行し、100 万人規模のユーザーに提供していました。停止が発生したのは、ヨーロッパと米国の全域がオンラインになるピーク トラフィック タイムでした。市場への発表から間髪をいれず、多くの新機能に「機能フラグ」を設定して非表示にする事態となりました。調査の過程で、ネットワーク層の重要な要素であるサービス間通信用 IP ポートにテレメトリが設定されていないことがわかりました。そこへ、想定外の要求が発生していたのです。図 1 に示すとおり、技術的観点から見れば、これは大規模な障害でした。この図から悲惨な状況がよくわかるかと思います。[1]

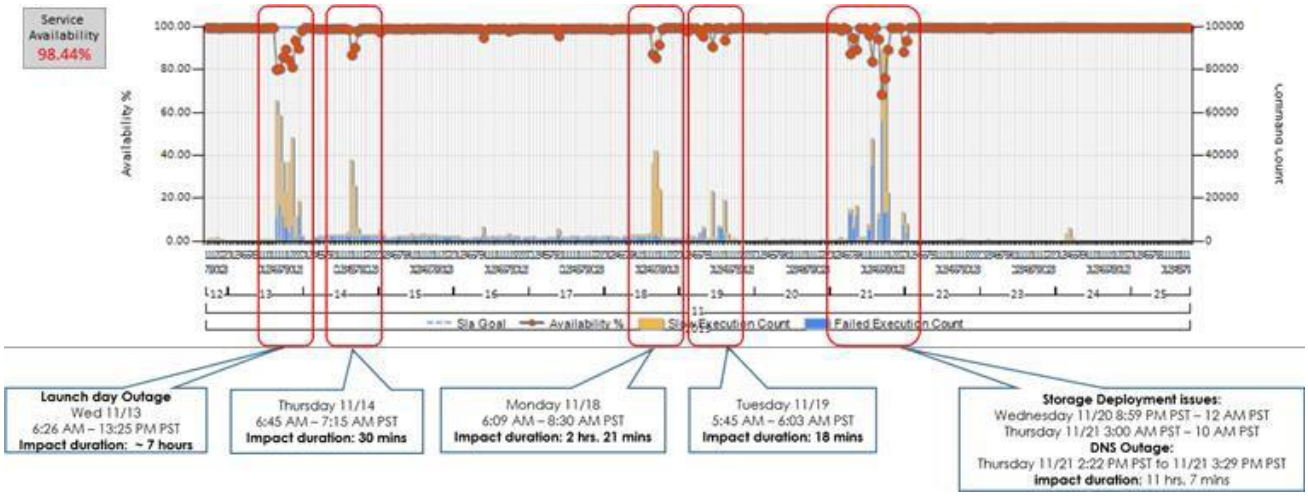


図1. VS 2013 のリリースは、VS Online サービスが連続して停止するという厳しい事態に見舞われた。

その反面、マーケティングの観点として見ると、リリースそのものは大成功でした。VSO はこの発表を転換点として、前月比 2 桁の急成長に向かい始めます (この成長は今も続いており、100 万人だったユーザーは翌年に 2 倍以上に増えました)。

フィーチャー関連とライブ サイト関連の作業調整

サービスの停止時、VSO はシカゴのデータセンターにある単一のスケール ユニットだけでホストされていました。しかしこの事態を受けて、今後は個別に展開した複数のスタンプへのスケールアウトが必要であると判断しました。ただ、これまで複数のスケール ユニットに移行する場合は、常にライブ サイトの作業よりもフィーチャー関連の作業を優先させていました。この優先順位が今回の件で変わりました。チームは、予定されていたフィーチャー作業を延期し、ライブ サイトの作業を優先することに決めたのです。VSO の更新プログラムを展開するにあたり、カナリア テストを実施する必要がありました。そこで、既存のシカゴのスケール ユニットには変更を加えず、展開の順序において既存のスケール ユニットの前にもう 1 つスケール ユニットを追加し、これを SU0 と名付けました。

新しいプロセスでは、まず、サン アントニオ (SU0) にスプリントを順次展開し、そこを作業場所として新しいリリースを数時間使用します。問題がなければ、図 2 に示すように展開を先へ進め、シカゴ (SU1) にロールアウトします。当然ながら、SU0 に問題があれば修復してやり直します。その後も時間をかけて、この流れに他の Azure データセンターをいくつか追加しました。2014 年秋には 5 つ目のデータセンターとしてアムステルダムを加えました。最近では、展開を国外へ拡張する一環として、オーストラリアを追加しています。各スプリントを世界各国にロールアウトする際のワークフローと自動化は、Visual Studio Release Management を使用して対応しています。

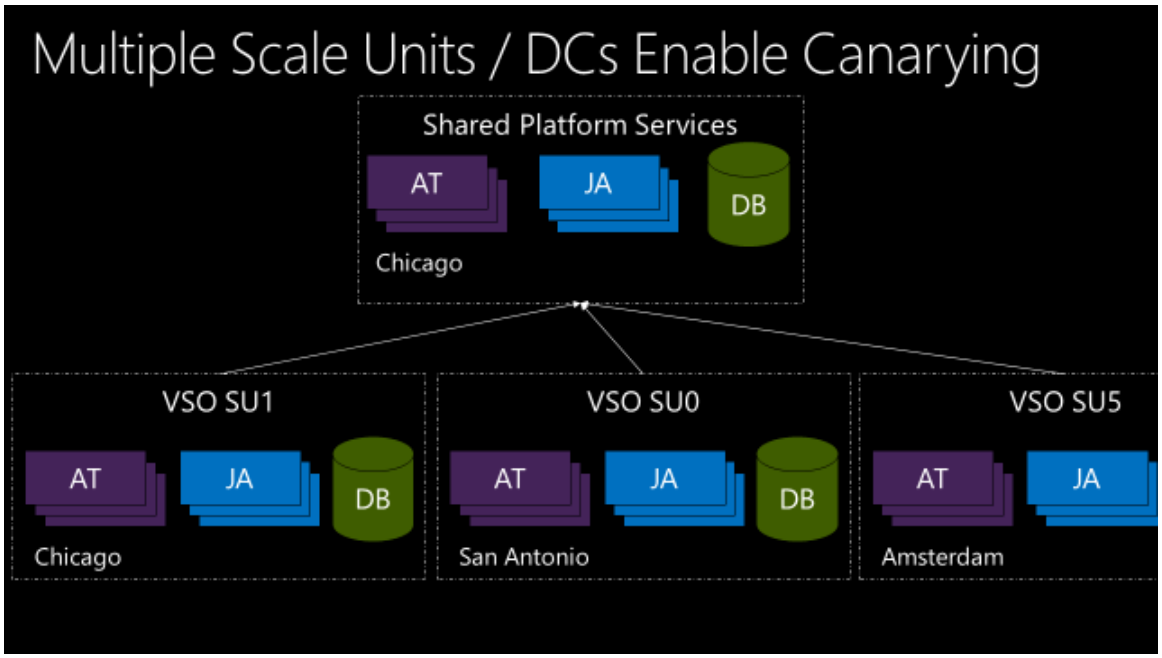


図2. 2013 年 11 月のサービス停止の後、VSO は単一のデータセンターから複数のデータセンターに拡張された。この措置により、展開の「カナリア テスト」とグローバルな拡張が同時に実現されている。

ライブ サイトのサービス品質に注力した効果は、結果にはっきりと表れました。2014 年の月次サービス レビュー (図 3) を見ると、2013 年 11 月のデータでは 43 件だったライブ サイト インシデント (LSI) が半年後に合計 7 件に減少しているのがわかります。しかも、(プレビュー版ではなく) 一般提供版サービスのインシデントはわずか 2 件です。それ以降も、DevOps への取り組みと主に運用面の健全性の強化に努めてきましたが、その道のりは決して平坦ではありませんでした。

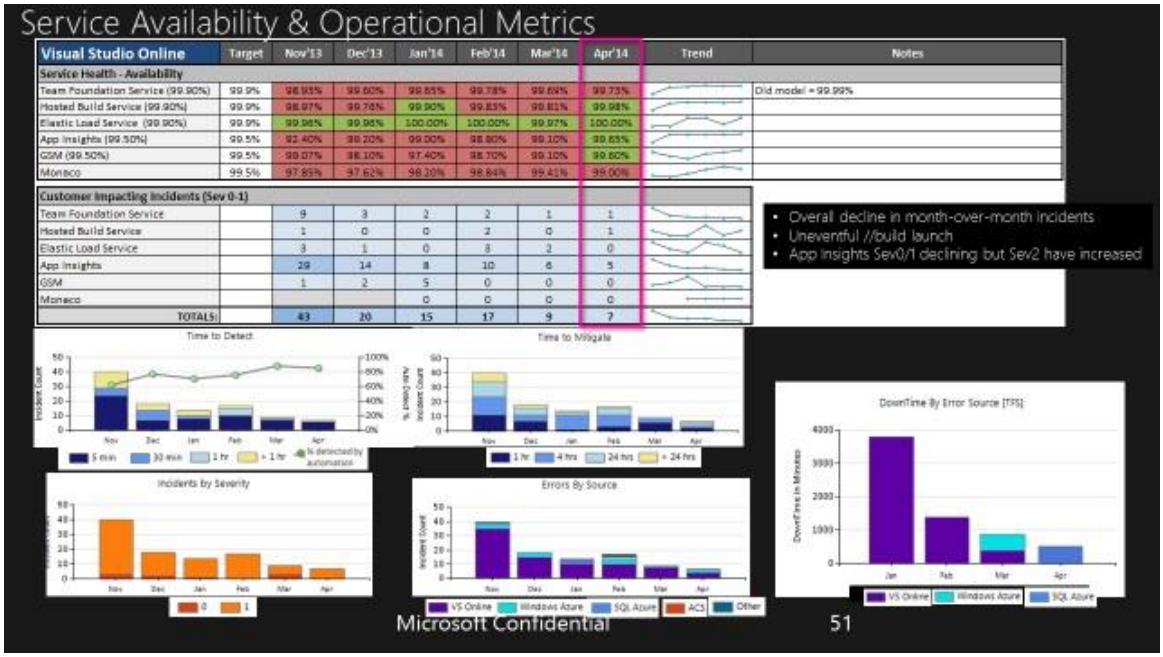


図3. VSO に起因する LSI は半年前に 43 件だったのに対して、2014 年 4 月期は 7 件のみ。しかも、その 7 件中、一般提供版のサービスに関するインシデントはわずか 2 件だった。

アジャイルから DevOps への移行

マイクロソフト デベロッパー部門 (DevDiv) は 7 年をかけてアジャイルを導入しました。この手法を堅実に実践したことで、XP が残した大きな技術的負債は 15 分の 1 に削減されました。また、部門内のすべてのメンバーを対象に、スクラム、多様な専門分野によるチーム構成、プロダクト オーナーシップについてトレーニングを実施しました。特に重視したのは、お客様への価値のフローです。VS2010 のリリース時には、製品ラインの顧客認知度が他を寄せ付けないレベルにまで高まっていました。[2]

VS2010 のリリースを終えた私たちを待っていたのは、Team Foundation Server を SaaS (サービスとしてのソフトウェア) として提供するための取り組みでした。現在は Visual Studio Online (VSO) と呼ばれるこの SaaS バージョンは、Microsoft Azure でホストすることになります。これを成功に導くために、DevOps 手法を導入する必要がありました。それは言わば、アジャイルから DevOps への開発手法の拡大です。この 2 つには、どのような違いがあるのでしょうか?

DevOps 文化の重要な要素は、実際の使用からの学びです。アジャイルでは、プロダクト オーナーが総責任者となり、バックログを適切に調整できることが暗黙の前提になっていました。これに対し、信頼性の高いサービスを提供している場合は、お客様が実際にどのように機能を使用しているかをほぼリアルタイムで観察できます。短いサイクルでリリースし、改良点を実験して評価し、お客様に変更についてどう感じたかを尋ねることが可能です。収集したデータは、次の改良の基盤になります。この観点で見れば、DevOps におけるプロダクト バックログは実際にはひとまとまりの仮説です。この仮説をもとにして実行中ソフトウェアで実験が行われ、継続的にフィードバックを得るという循環ができあがります。

図 4 に示すように、DevOps は 4 つの流れをベースにしてアジャイルから発展しました。

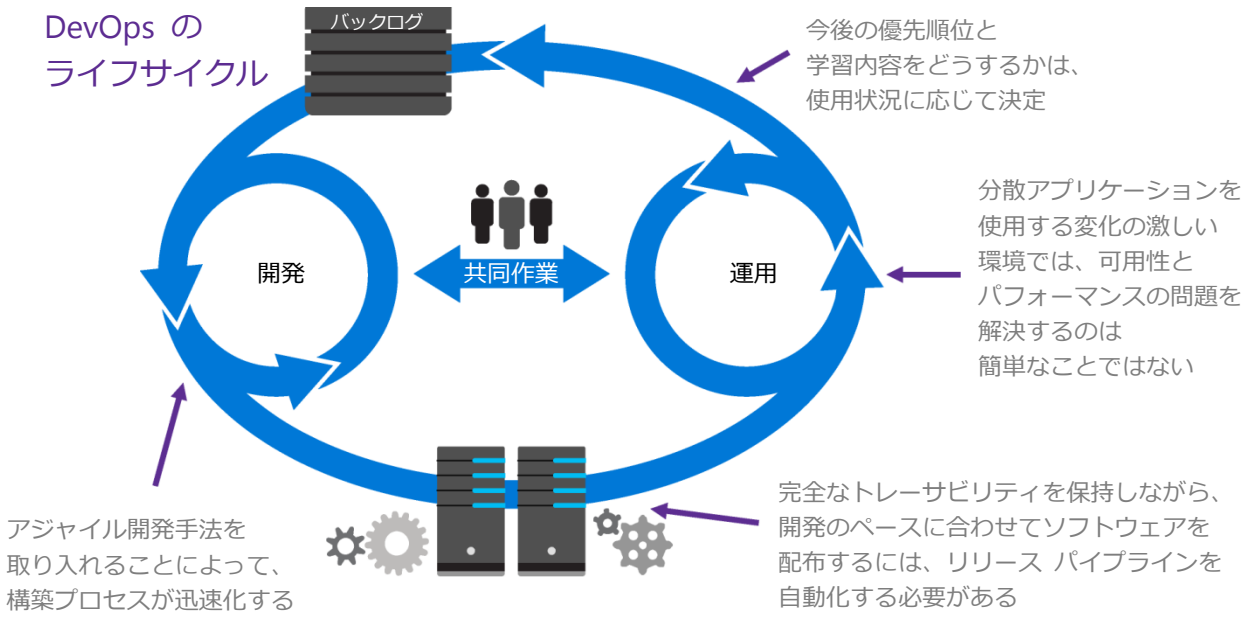


図4. 4 つの流れは、DevOps への発展を後押しする要素として捉えることができる。

「クラウド生まれ」の多くの企業とは異なり、マイクロソフトは SaaS サービスから始まったわけではありません。当社のほとんどのお客様は、マイクロソフト ソフトウェアのオンプレミス バージョンを使用しています (Team Foundation Server は 2005 年に初めてリリースされ、現在提供されているバージョンは 2015 です)。VSO を開始するにあたり、私たちは製品の SaaS バージョンと「パッケージ」バージョンの両方に対してシングル コード ベースを維持しつつ、クラウド ファーストで開発を進めることを決めました。エンジニアがコードをプッシュすると、継続的な統合パイプラインがトリガーされます。図 5 に示すように、3 週間のスプリントが終わるごとにクラウド版をリリースし、4、5 回スプリントを繰り返した後でオンプレミス製品向けに四半期単位の更新プログラムをリリースします。

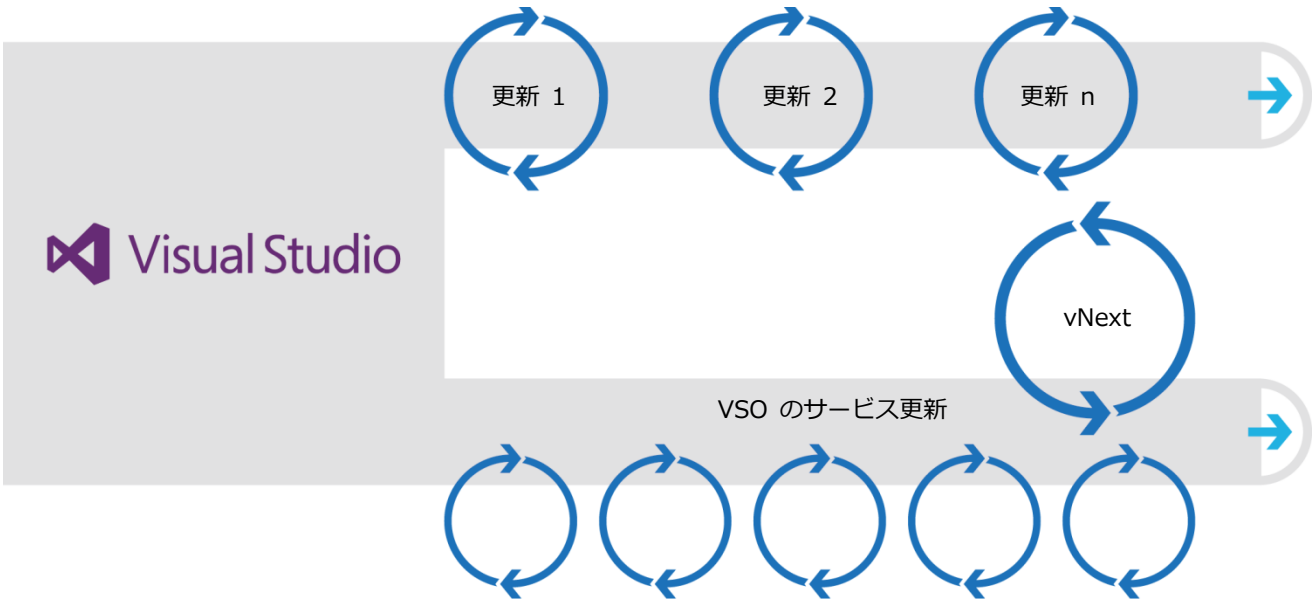


図5. VSO (SaaS 製品) と Team Foundation Server (TFS、オンプレミス製品) に対してシングル コード ベースを維持。3 週間ごとに新しいフィーチャーが VSO にライブ展開される。このフィーチャーが四半期ごとに TFS の更新プログラムにロールアップされる。2015 など、TFS メジャー アップデートのリリースは最新の VSO をベースに行われる。

作業公開の管理

短いサイクルでのリリースは、サービスの開発に大きなメリットをもたらします。マイクロソフトの場合、リリースは 3 週間のスプリントが終わるごとに実施されます。この間隔で作業成果を公開することには良い効果があります。ただし同時に、公開のタイミングを管理する必要も生じます。これには次のような問題が伴います。

- 複数のスプリントをまたぐフィーチャーをどのように作業するか。
- 後で変更される可能性があるフィーチャーを使用してもらってフィードバックを得るために、どのような方法でフィーチャーを実験するか。
- 公開や市場投入の準備が整う前にサービスや機能を導入する「ダーク ローンチ」をどのように行うか。

マイクロソフトでは、これらのすべてを解決するために、機能フラグ パターンを使用しています。機能フラグは、ユーザーまたはユーザー グループに対するフィーチャーの運用公開を管理するしくみです。新しいフィーチャーを作業しているチームは、機能フラグ サービスにフラグを登録します。既定ではフラグはオフになっています。自分の作業成果を他の人に試してもらう準備が整ったら、相手の ID に対して必要な間だけフラグをオンにして運用状態にします。フィーチャーの修正が必要になればフラグをオフにし、再展開を実施しなければフィーチャーは非公開状態になります。

機能フラグを使用すると段階的な公開管理が可能になるため、運用環境でのテストにも利用できます。新しい機能の公開は通常、まず社内に対して行い、次にアーリー アダプターに対して、その後さらに広範囲のお客様に対して実施します。パフォーマンスと使用状況を監視することで、新しいサービス コンポーネントに大きな問題がないことを確認できます。

コードのベロシティと分岐

マイクロソフトが 2008 年に初めてアジャイルに移行した当時、私たちは適切なクオリティ ゲートと分岐構造を使用すれば、コードの品質を向上できると考えていました。導入当初、開発者はかなり複雑な分岐構造で作業し、厳格な完了の定義を満たしたコードしかプロモートできませんでした。ゲートチェックインも採用されており、トランクから効率よく「最新版を取得」し、新しい変更セットでシステムをビルドして、ビルド ポリシーを適用していました。

しかし、このような分岐構造には予期せぬ結果が伴いました。リーフ ノードからトランクへ向かうコードの流れに、何日もの、時には何か月にも及ぶ停滞が生じ、マージ前のコードがブランチに長く留まり出したのです。これは、重大なマージの負債でした。作業成果をマージする準備が整うとトランクの流れは一気に前進するものの、マージの競合が大量に発生し、その調整に多くの時間が費やされ、無駄も少なくありませんでした。

そこで、2010 年までに実施した最初の措置は、分岐構造を思い切ってフラットにすることでした。現在は分岐がほとんどなく、あっても多くは一時的なものです。また、コードの流れを最適化するという目標を明確に設定しました。具体的には、コードのチェックインから変更セットを作業中のすべての開発者に公開するまでの時間を最小限まで短縮することを目指しました。

次の措置では、Git を使用した分散バージョン管理への移行を実施しました。Git は現在 VSO と TFS でサポートされています。お客様や社内の他部門のほとんどは今でも一元管理型のバージョン管理を使用していますので、VSO と TFS ではどちらのモデルもサポートしています。Git には、一時的な分岐を手軽に使用できるという利点があります。1 つの作業項目についてトピック ブランチを作成し、変更をメイン ラインにマージし終えたらトピック ブランチを削除できます。

コミットされたコードはすべて Master (トランク) に入ります。プル リクエストのワークフローは、コード レビューとポリシー ゲートの両方が組み合わされたものです。このプル リクエストによって、流れを止めずに小規模なブランチで簡単にマージを実行でき、すべての開発者がコードを把握しやすくなります。このプロセスでは、開発者の作業は短期間だけ分離され、互いに独立して行われた後、停滞を生じずに統合されます。ブランチはブックキーピングのオーバーヘッドがなく、不要になれば縮小されます。

強化版アジャイル

マイクロソフトでは現在もスクラムを実践していますが、コミュニケーションをとりやすく、地域を越えて拡張できるように、本質だけを残してスリム化しています。たとえば、メインで作業にあたるのはフィーチャー クルーで、これはスクラム チームに相当します。プロダクト オーナーがチーム内に配置され、毎日作業に参加します。プロダクト オーナーとエンジニアリング リードが共同でチームの代表を務めます。

マイクロソフトでは、チームの自立性と組織としての連携を指針に掲げています。フィーチャー クルーにはある種の化学反応が生まれています。チームは多様な専門分野で構成されているうえに、昇進コースを短縮して、開発者とテスターを共通のエンジニアリング担当者に進化させました (これが大きな士気高揚につながっています)。

チームの結束は強く、8 ～ 12 名のエンジニアで 1 つのクルーが構成されています。最低でも 12 ～ 18 か月間は協力して作業にあたるようにしており、多くの場合はさらに長い期間チームとして作業します。作業調整に問題がある場合は、クルー間でエンジニアを移動させるのではなく、別のクルーにバックログ項目を多めに引き受けてもらいます。

フィーチャー クルーは全員が図 6 のようなチーム ルームで作業します。これは広いオープン スペースの一画ではなくチーム専用の部屋です。共通の領域を作業するメンバーがスペースを共有しているため、気兼ねなく会話ができます。チーム ルームの周囲には、ブレイクアウト用の小さいフォーカス ルームがいくつかありますが、議題や制約のない会話はチーム ルームで行われています。スタンドアップ ミーティングの際は、全員が立って打ち合わせをします。

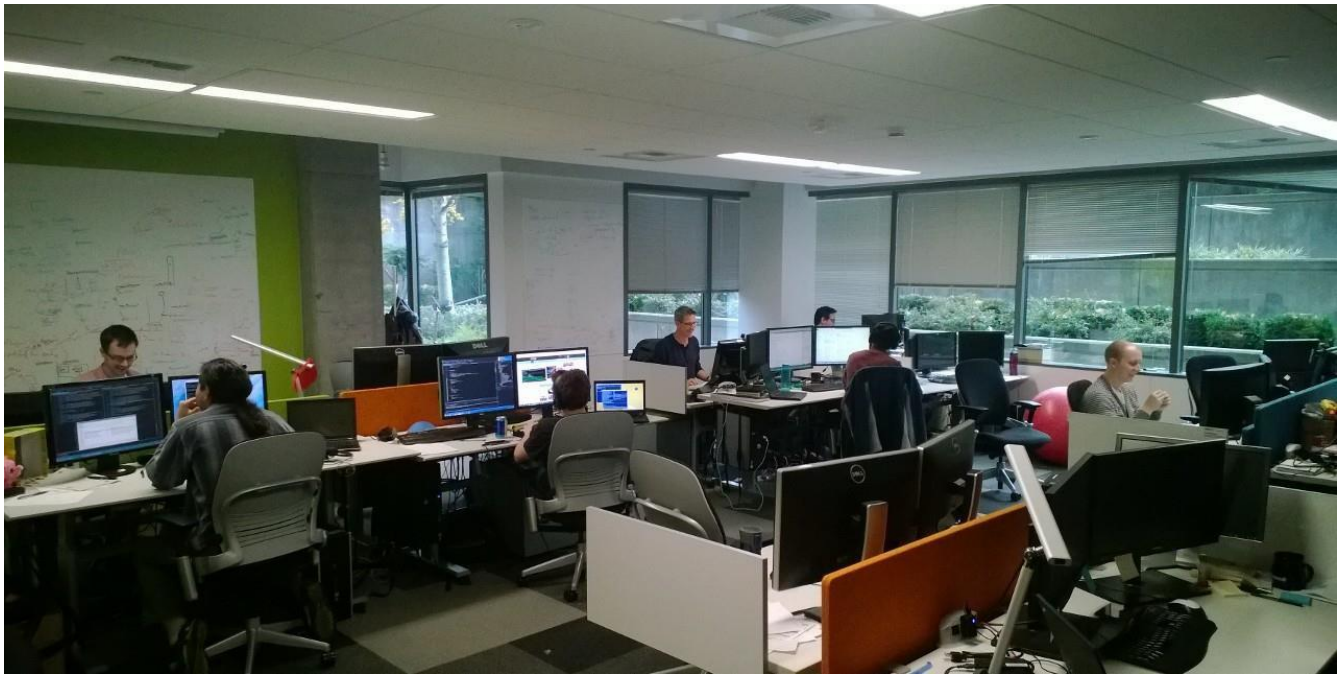


図6. フィーチャー クルーは全員が同じチーム ルームで作業する。

「3 週間」というスプリントの長さには、経験の末たどり着きました。これよりも期間が短いと、十分な価値を提供したり世界各地の拠点間で調整を図ったりすることが難しくなりました。反対にそれ以上長いと、ダーク ローンチ状態の作業が増えすぎてしまいました。マイクロソフト内には、2 週間のスプリントを採用しているグループもあれば、スプリントの時間枠を設けずに 1 日に何度も新しい作業成果に対する実験のフライティングを実施して効果をあげているグループもあります。

完了の定義は概念的にはとてもシンプルです。ビルドし、実行するだけです。コードはスプリントの最後に何百万ものユーザーにライブ展開されます。ライブ サイトにインシデントが発生すれば、担当者 (またはメンバー全員) はすぐに把握し、根本原因を解決します。

スクラム マスターの役割は持ち回りで担当し、全員がスクラムを運営する責任を経験するようにしています。外部への伝達では、スプリントの形式張った面はできるだけ控えめにしています。スプリントの開始時、クルーはプロダクト バックログから作業を取得し、1 ページに収まる電子メールでスプリント計画を伝達します。メールには VSO 内のプロダクト バックログ項目に対するハイパーリンクが記載されます。スプリント レビューは、このメールに対する更新として、約 3 分のビデオを追加して配布されます。ビデオの内容は、そのスプリントにおけるチームの作業成果で何ができるようになったかをお客様の視点で示すデモです。

また、計画作成でも大げさにならないよう心がけています。通常は 18 か月単位のビジョンを掲げ、これを絶対的な目標としています。ビジョンは状況に応じて、テクノロジ スパイク、ストーリーボード、概念ビデオ、概要ドキュメントに反映されます。また、6 か月を 1 つのシーズン (「スプリング」または「フォール」) として、この期間にチーム間のコミットメントと依存関係の強化を図っています。3 週間のスプリントが終わるごとに、各フィーチャー クルーのリーダーが「フィーチャー チャット」に集まり、意図したスタック順位の共有、軌道修正の確認、情報交換、他チームとの調整などを行います。たとえば、ライブ サイト作業やユーザーに表示するフィーチャーのランク付けの変更がフィーチャー チャットで行われることがあります。

ビルド、評価、学び

従来のアジャイル手法では、プロダクト オーナーがプロダクト バックログ項目 (PBI) のランク付けを行い、PBI はおおむね要件として扱われます。PBI はユーザー ストーリーの形で記述されることもあれば、簡単な形式で作成される場合もありますが、いずれにしても決定事項であることに変わりはありません。マイクロソフトでも以前はこの方法で作業していましたが、後に、柔軟性と効率に優れたアプローチを開発しました。

現在は PBI を DevOps の手法に合わせて仮説として捉えています。仮説ですから、実験に発展させる必要があります。実験を通して、仮説を支持する証拠または反証が得られ、それらの証拠から検証済みの学びが得られます。[3]

良い実験を行うには、統計的に有意なサンプルと比較のための対照グループが必要です。また、結果に影響を与える要素の理解が欠かせません。図 7 ～ 11 で示した状況を例に考えてみましょう。VSO の新規ユーザーがすぐにチーム プロジェクトを作成しないことが問題点です。チーム プロジェクトを作成しないと、できることは限られてしまいます。図 7 は Web 版のスタート画面です。

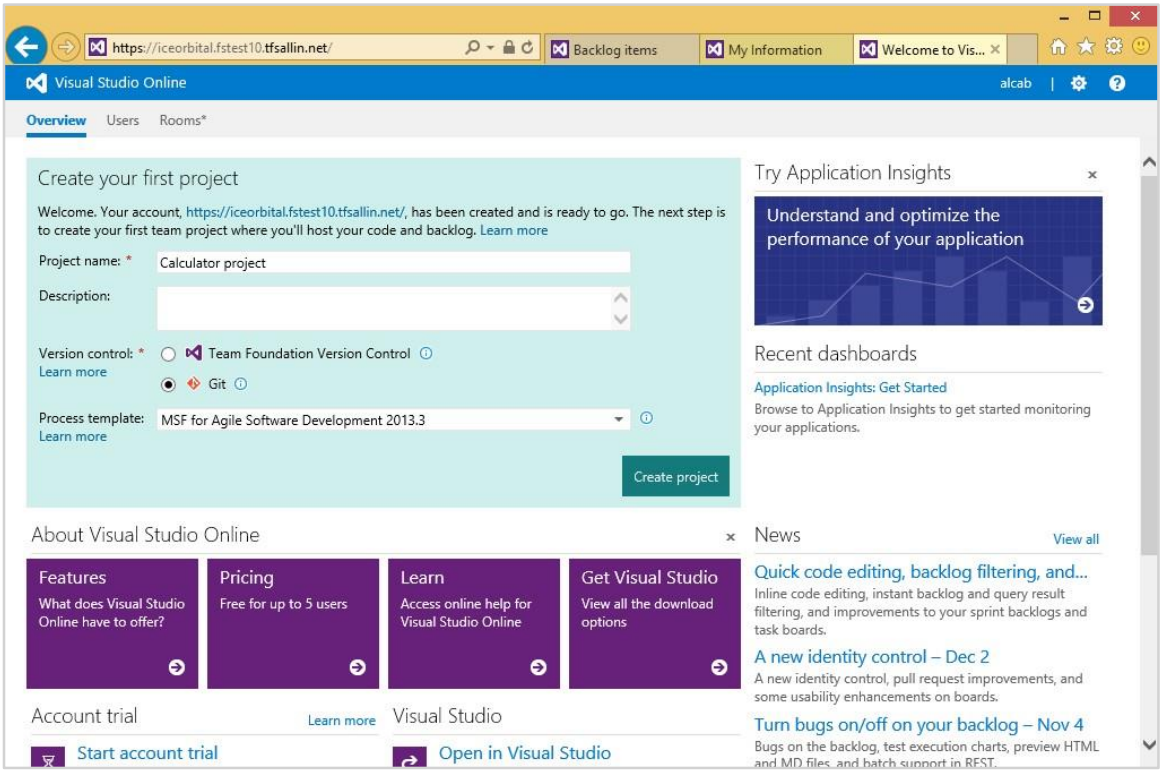


図7. 以前のエクスペリエンスでは、アクションや不要な要素が多すぎて、お客様がプロジェクト作成という次のステップに自然に進むことができなかった。

Web 版と IDE 版の両方のエクスペリエンスに修正を加えました。図 8 は IDE 版の新しいエクスペリエンスを示しています。ユーザーが 2 つの画面で VSO のサインアップとプロジェクトの作成を完了できるようにしました。この新しいエクスペリエンスは、6 つのスケール ユニットのうち 2 つを使用して実験として実装しました。

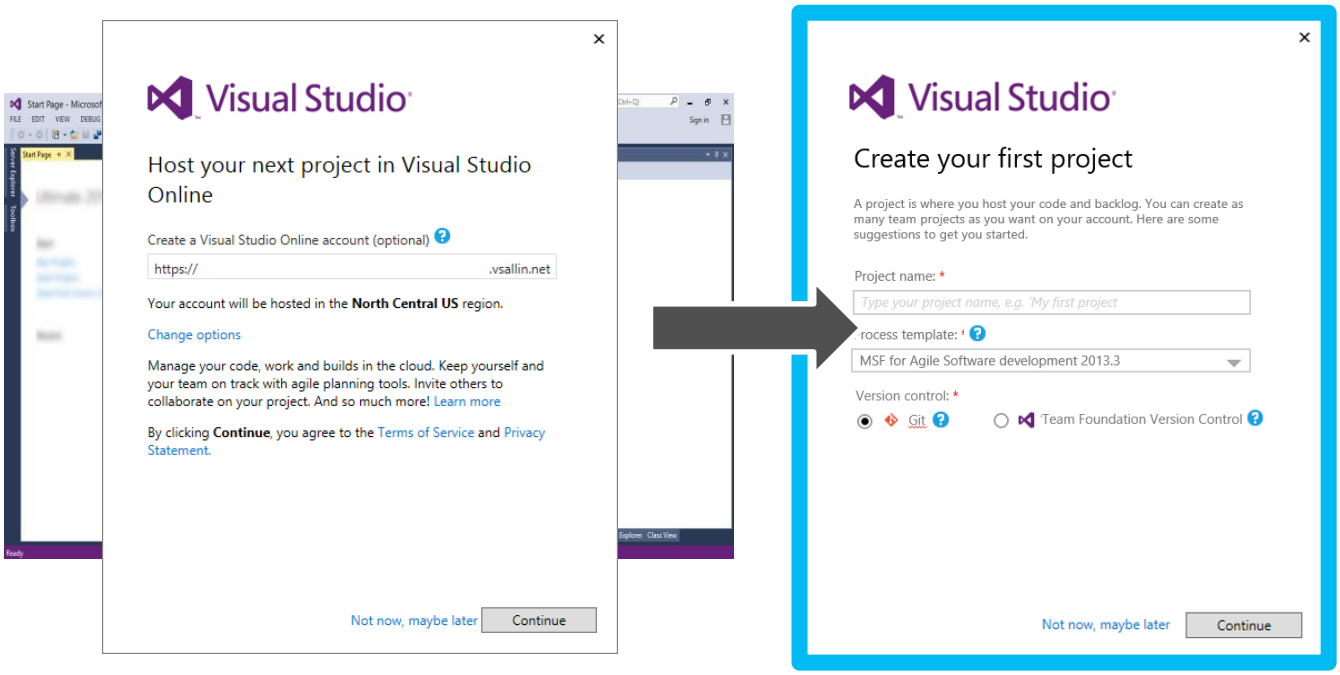
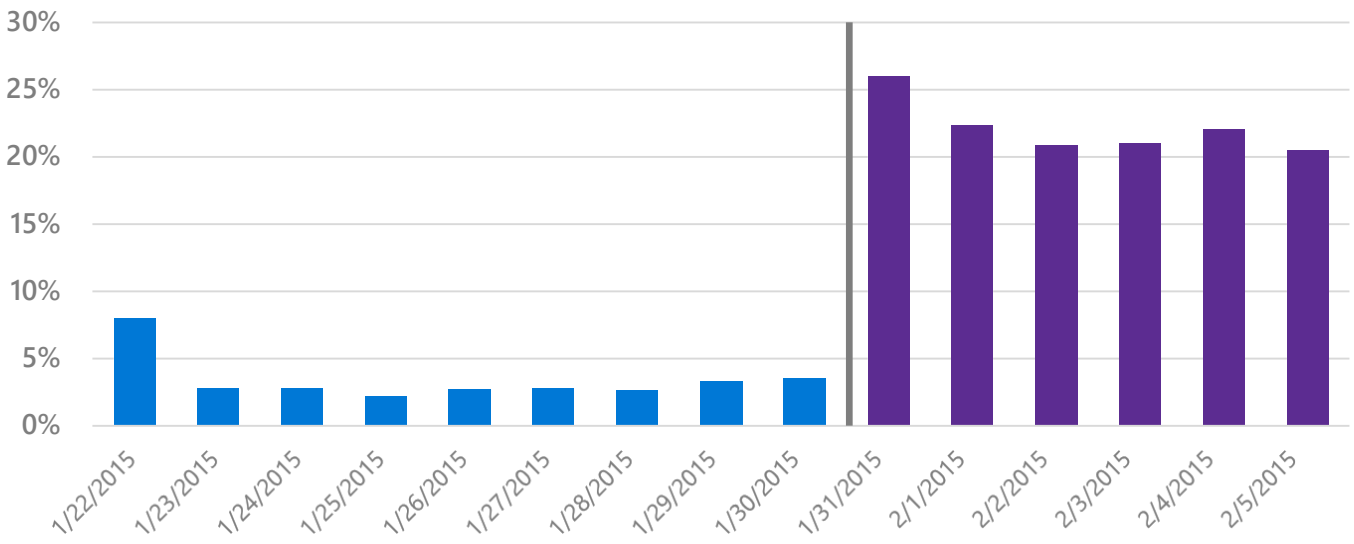


図8. プロジェクト作成に焦点を絞った IDE 版フローのエクスペリエンスは 2 つのスケール ユニットで実装された。

図 9 に示すのは IDE 版の実験の効果です。新規アカウント作成時のチーム プロジェクト登録は 3% から 20% に増えました (7 倍の向上)。

アカウント作成日にプロジェクトを作成したアカウントの割合 (1 月 22 日～ 2 月 5 日) IDE 版のみ



変更は SU4 (ヨーロッパ) と SU5 (米国中南部) で行われた。

図9. IDE 版フローの変更により、IDE 版で実行された新規アカウントの作成は 3% から 20% に急上昇した。

私たちは同時に Web 版の登録にも変更を加えましたが、こちらは結果のパターンが異なっていました。図 10 は Web 版の登録に加えた変更を示しています。

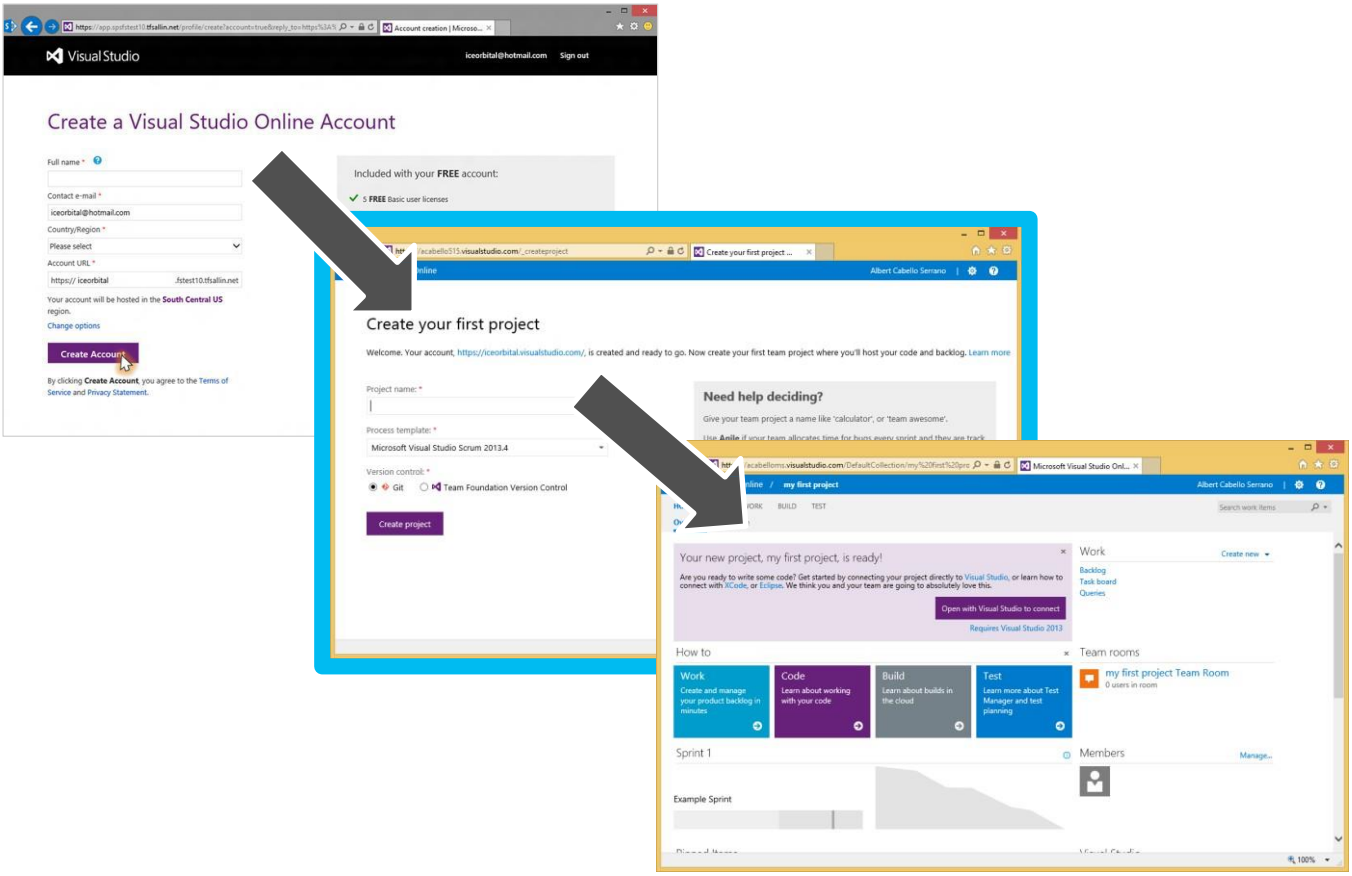


図10. Web 版の登録も同様に簡略化を施し、2 つ目の画面をプロジェクト作成専用にしました。

私たちは、アカウント作成日にチーム プロジェクトを作成したユーザーの数を測定しました。図 11 の紺色から紫への棒グラフの変化が示すとおり、数値はおよそ 18% から 30% (1.7 倍) に向上しました。悪くはありませんが、期待値と IDE 版の結果のどちらと比べても、これは思いがけないほど低い数字でした。

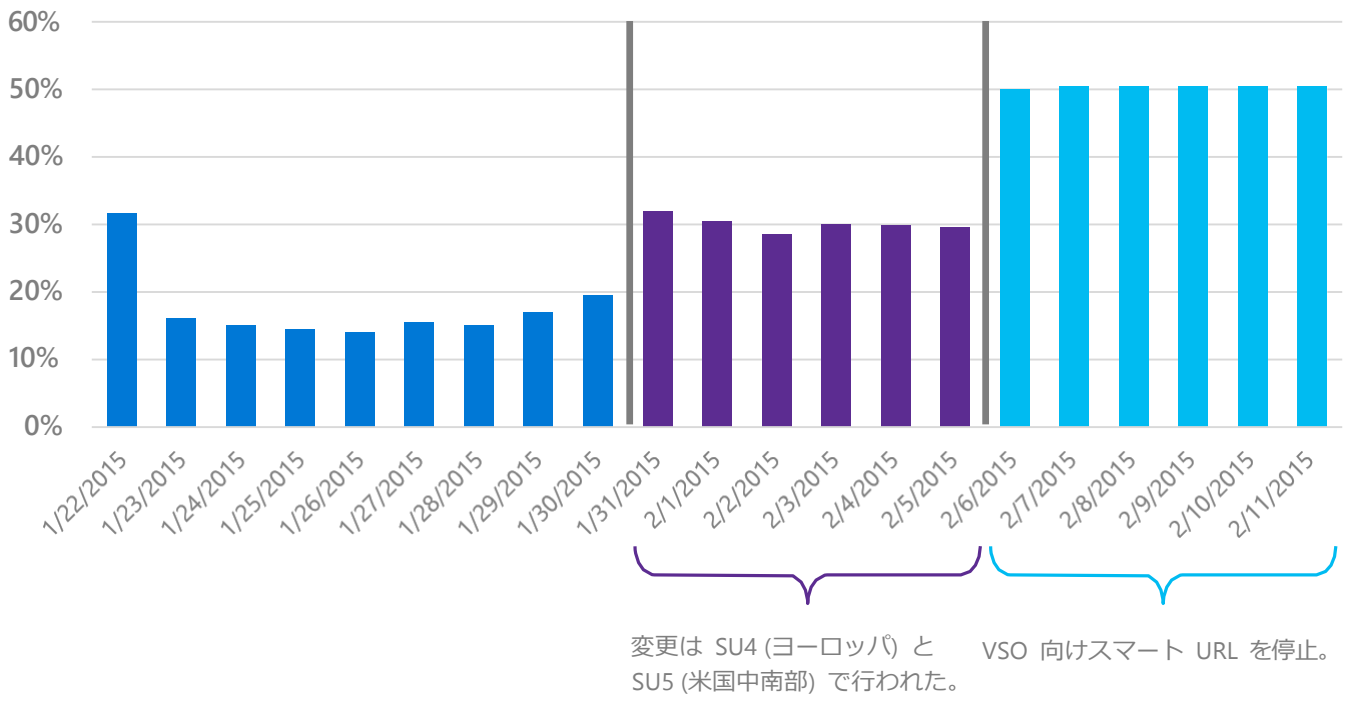


図11. Web 版のアカウント作成の結果は、2 つの実験の予期せぬ相互作用が原因で不正確なものとなった。一方を停止すると、メインの結果がはっきりと確認できた。

調査したところ、別の実験が行われていることがわかりました。そちらの実験では、じょうごの入口で条件に当てはまらないユーザーにもアカウントへの登録を促し、結果として多数のユーザーが応じないということが起こっていました。これは、2 つの実験が相互干渉するという不適切な対照グループの例でした。アカウント作成の促進を停止すると、青いバーが示すとおり、プロジェクトの作成は 50% (2.8 倍。IDE 版の 2.5 倍) へと飛躍的に向上しました。

私たちは実験から 1 つの教訓を学びました。実験結果が有効かつ反復可能で正しく制御されたものであるかどうかは、すぐに明らかになるわけではないということです。

DevOps に対応したエンジニアリング システム

VSO はグローバル サービスです。24 時間年中無休で提供され、99.9% の信頼性が返金規定で保証されています。このサービス レベル アグリーメント (SLA) は目標ではありません。達成できなければ返金が適用される最低ラインです。VSO で目指しているのは 100% の可用性と 100% の顧客満足度です。メンテナンスのダウンタイムがなく、クリーン インストールもなし、あるのは更新プログラムのみで、すべてが自動化されていなければなりません。これを達成するには、すべてのサービスを互いに分離し、あいまいな余地のない契約を提示し、明確なバージョン管理を用意する必要があります。お客様の環境と同様、VSO は Azure パブリック クラウドで実行されるため、インフラストラクチャは Azure で制御されます。

展開の自動化

展開とサイトの信頼性に関する作業は、エンジニアリング チーム直属の「サービス デリバリ (SD)」と呼ばれる小規模なチームによって管理されています。SD は各スプリント終了後の翌月曜日に展開を開始します。展開は、図 12 に示す Visual Studio Release Management を使用して自動化されています。インシデントの調査や修復が必要になった場合に備えて、展開は営業時間中に実施されます。VSO には顧客データが格納され、データベース スキーマは頻繁に更新されるため、ロールバックを行うことはなく、ロールフォワードだけが実施されます。自動化はスケール ユニット単位で順次ロールアウトされます。最初は SU0 でカナリア テストを実施します。このアプローチにより、公開を制御しながら段階的に行い、正常性とユーザー エクスペリエンスを確認してから次へ進むことができます。

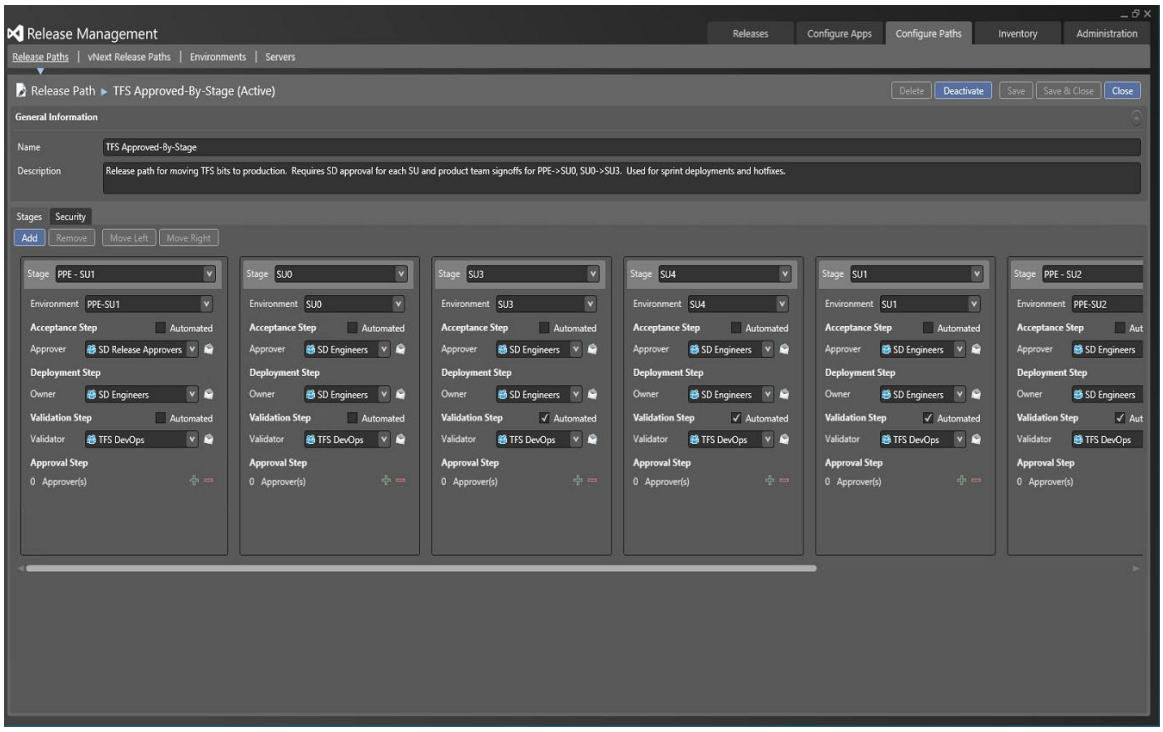


図12. VS Release Management を通じてスケール ユニットへの順次展開を制御。

すべての展開が、まず SU0 で数時間実行されてから、次のスケール ユニットへ進められます。問題があれば、SU0 の段階で特定し、展開を続行する前に根本原因を解決します。データが破損していて障害復旧が必要な場合には (幸いにもこれまでに 1 度しか発生していませんが)、SUO の時点で実施します。その際は、お客様のデータではなくマイクロソフトのデータが使用されます。

テレメトリ

VSO のコードの中で最も重要なものは、間違いなくテレメトリです。サービスに問題が発生しても、監視を利用不能にするわけにはいきません。サービスに問題が生じれば、監視アラートとダッシュボードで直ちに報告する必要があります。図 13 は、サービス正常性の概要の例です。

図13. VSO サービスのテレメトリを提供する多数のトップレベル グラフの 1 つ。

VSO のすべての要素が、可用性、パフォーマンス、使用状況、トラブルシューティングをあらゆる角度から把握できるように実装されています。マイクロソフトでは、テレメトリに対して次の 3 つの原則を適用しています。

あらゆる情報を徹底的に収集する。

自己満足のメトリックを排除し、実用的なメトリックに的を絞るよう努める。

実施した測定の効果を確認できるように、入力値そのものではなく結果と比率を評価する。

1 日に収集されるテレメトリ データは、60 ～ 150 GB にのぼります。アカウント レベルで詳細に分析することもできますが、マイクロソフトのプライバシー ポリシーにより、お客様が詳細情報の開示を選択していない限り、データは匿名化されています。収集しているデータには次のようなものがあります。

動作状況のログ。 VSO サービスへの Web 要求に関するデータをすべて収集します。このデータをもとに、実行時間を追跡し、各コマンド数を計測して、特定の呼び出しや依存サービスの処理速度が低下していないか、再試行の頻度が多すぎないかを判断します。

トレース。 エラーが発生するとスタック トレースが開始され、それをもとにエラー発生後の呼び出しシーケンスのデバッグを実行します。

ジョブ履歴。 ジョブとは、サービス全体のアクティビティを 1 つにまとめたワークフローです。

パフォーマンス カウンター。 パフォーマンスに関するデバッグの経験がある方にはおなじみのカウンターです。システム リソースが正常な状態にあるかどうかを追跡します。VSO では 1 日あたり約 5 千万件のイベントが生成されます。

Ping メッシュ。これは、ネットワーク基盤層を瞬時に視覚化したもので、接続の可用性を世界規模で確認できます。

代理トランザクション。これは「アウトサイドイン テスト」とも呼ばれ、グローバル サービス モニタリングで実行されます。世界各地のアクセス ポイントから正常性が報告されます。

お客様の使用状況。使用状況に関しては、「玄関口」、コンバージョンのじょうご、エンゲージメント、上位顧客を測定します。

KPI メトリック。これは、サービスのビジネス面の正常性を測定するためにテレメトリ システムで集計されるメトリックです。

追跡

当然のことですが、どんな要素も軽く扱うことはしません。すべてのメンバーが「ライブ サイトの文化」の中でトレーニングを受けています。ここでは、ライブ サービスの状態が常に最優先です。ライブ サイトに問題が発生すると、他のどの作業よりも問題の検出と修復が優先して行われます。廊下の至る所にライブ ダッシュボードが設置されているため、だれもが正常性や障害に気付くことができます。

ライブ サイト インシデント (LSI) は、すべて VSO に記録され、必ず根本原因が解決されて、週ごとにレビューが行われます。図 14 は LSI の一例です。

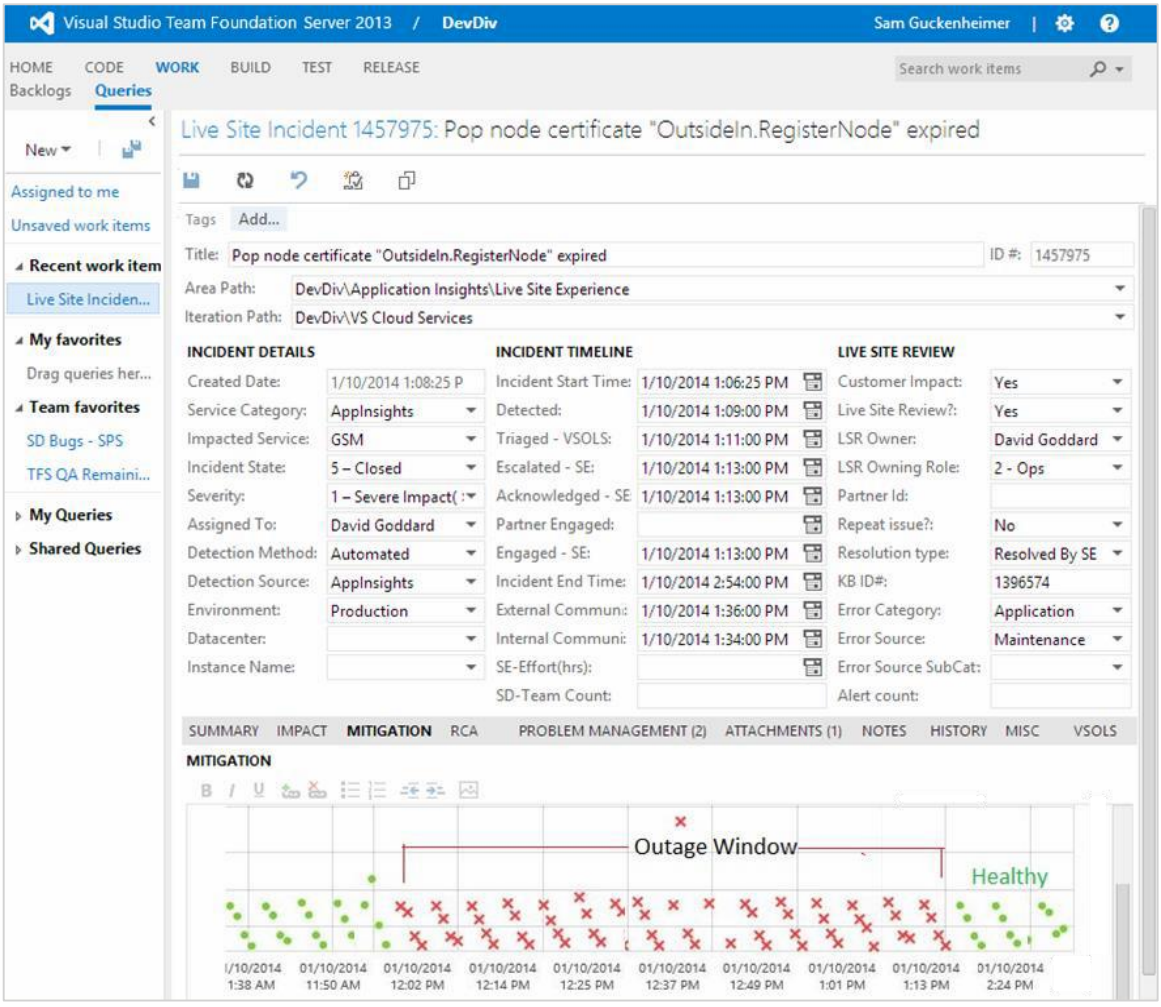


図14. LSI では、運用サービスで発生したすべてのインシデントが追跡される。根本原因の解決に関する詳細も含まれる。

群衆の知

DevOps でよくある質問は、「だれが呼び出しに対応するのか?」というものです。マイクロソフトでは、サービス デリバリ (SD) チームが 24 時間体制で 1 次窓口を務めています。チームのメンバーは世界各地に配置されています。開発チームへのエスカレーションが必要になると、SD チームはフィーチャー クルーに所属する対応担当者 (DRI) の時間刻みのスケジュールを確認します。マイクロソフトのサービス レベルの期待値には、営業時間中は 5 分で、営業時間外は 15 分で DRI がライブ サイトの「指揮所」に詰めること (電話会議に出席するなど) が設定されています。

マイクロソフトが「問題の修復」をどのように考えているかという点も重要です。マイクロソフトにおける問題の修復とは、VM のバウンスではなく、根本原因の解決です。その理念は、トヨタの工場で実践されている「アンドンの紐」に似ています。「アンドンの紐」とは、生産ラインで欠陥のある部品が見つかったときにどの作業者でも引けるように垂れ下がった紐のことです。[4] マイクロソフトでも再発防止のために発生現場でコードや構成を修正し、テストを強化して、パイプラインをリリースします。その後、ライブ サイトのすべてのメトリックに対して分単位で改善状態を測定します。

これまでの成果の中で最も大きなものは、人の手によってエスカレートしなくても DRI にオートダイヤルできるようにアラートを調整して精度を高くしたことです。これを実現するために、図 15 に示すような正常性モデルを作成してノイズの多い冗長なアラートを取り除き、アクションが必要な状況を示す的確な境界値を設定しています。この措置により、アラートの精度は 40 倍にも向上しており、2015 年 2 月時点で P0 および P1 のアラートは 1 つの例外もなく正確にルーティングされています。

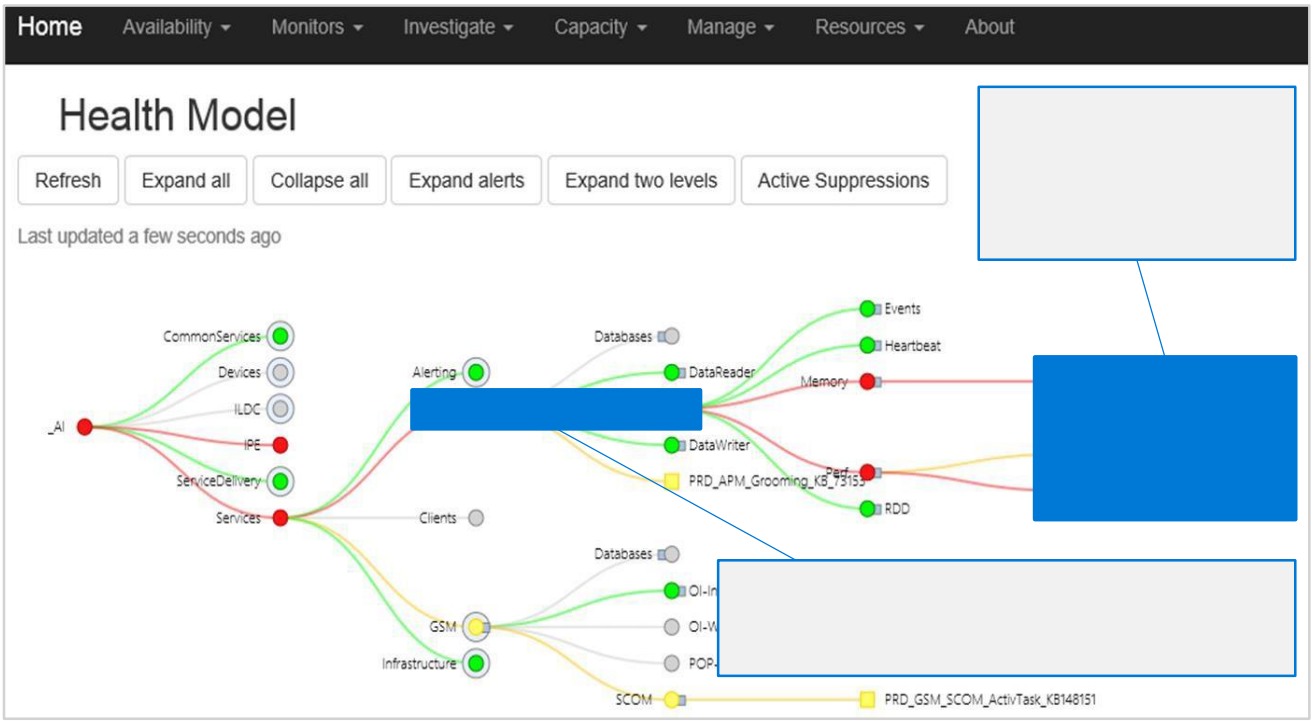


図15. 正常性モデルによって重複したアラートが自動的に排除され、どのフィーチャー エリアに根本原因の疑いがあるかが特定される。

ユーザー エクスペリエンスからの学び

私たちはこれまで、人間工学のあらゆる面からユーザー エクスペリエンスをきわめようと努めてきました。時には、そのためにかえって業務が困難になることもあります。その良い例は、99.9% のサービス レベル アグリーメント (SLA) の計算方法です。

私たちが目指す目標は SLA ではなく、満足度 100% です。これは、外れ値である 0.001% のケースを気にかける必要があることを意味します。外れ値はややもすると平均の中に埋もれてしまうため、マイクロソフトでは 3 世代のアルゴリズムを経て SLA 計算方法の精度を高めてきました。図 16 は、それらのアルゴリズムを並べて示したものです。最初は、図中に点線で表されているアウトサイドイン モニタリングを使用して、サーバーの可用性を追跡しました。2 つ目に使用したアルゴリズムでは、コマンド実行の合計数に対する応答の遅いコマンド数と失敗したコマンド数に注目し、外れ値の大きいケースの問題点を特定しようとしました (図中の黄色と青色の線)。サービスが拡大するにつれて分母が大きくなり、お客様が低品質だと感じたかもしれない多くのケースが目立たなくなってしまいました。現在使用している計算方法は、ユーザー分数の合計に対するサービスを利用できなかったユーザー分数の比率の算出です (黒線)。

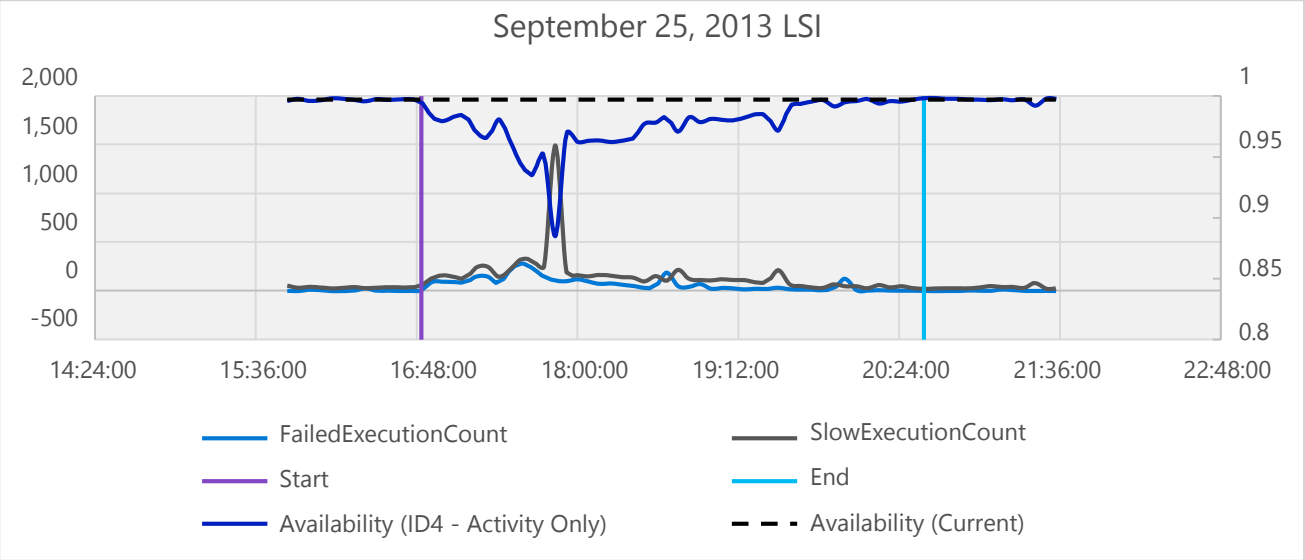


図16. この LSI は、SLA の 3 つの計算方法を比較したもの。アウトサイドイン モニタリングでは問題が発見されなかった。コマンド パフォーマンスの追跡では 1 時間のパフォーマンス低下が見られたのに対し、リアル ユーザー監視では約 3 時間の応答遅延が見られた。

ご覧のとおり、最初 (点線) の SLA 計算は、4 時間の期間中に問題がなかったことを示しています。2 つ目 (黒線) は約 1 時間のパフォーマンス低下があったことを示し、3 つ目は約 3 時間のパフォーマンス低下があったことを示しています。契約上の「義務」はありませんが、私たちは自社のサービスに対する測定を厳しくしています。ここは、マイクロソフトが絶えず向上する必要のある領域です。

セキュリティ

データのプライバシー、データ保護、サービスの可用性、サービスのセキュリティは、通常のセキュリティ対策の一環として確保に努めています。以前は、オンプレミスのソフトウェアに対して、セキュリティ侵害を阻止するために徹底した防御を構築しようと多大な労力を投じていました。これは今でも変わりませんが、異なる点としては、現在はサービスが既に浸透しているため、進行中の侵害を検出する必要があるという前提が出発点になっています。[5]

SU0 を展開用のカナリアとして使用することに加え、ちょうど火災訓練を行うようにこのデータセンターでセキュリティ訓練を実施し、VSO サービスの強化に努めています。有料のお客様に影響を与えずに運用環境への攻撃を実行できるスケール ユニットがあることで、大きなメリットが得られています。

新しい展開パターン

オンプレミス ソフトウェアのみを生産していたときは、平均故障間隔が最小限になるようにエンジニアリング システムとプロセスを調整していました。これは裏を返すと、お客様にソフトウェアを出荷した後で、修正が必要なエラーが発生すると、修復コストが高額になることを意味しました。そのため、新バージョンの再リリースという事態を避けるために、あらゆる手を尽くして起こりうる問題を事前に予測していました。

DevOps の考え方は、これとは逆です。手間をかけずに再展開でき、修復コストを最小限に抑えることができれば、目指す目標は違うものになります。必要となるのは、サイクル時間、つまり、検出、修復、学習に要する時間の最小化です。

強い復元力を実現するための設計パターンも複数考えることができます。依存関係には障害が付きもので、特にトラフィック ピーク時に発生しやすくなっています。その場合、サービスのパフォーマンスを安全に低下させ、「ブレーカー」の利用 [6] などの再試行戦略を十分に検討し、Netflix で話題となった Chaos Monkey[7] のような手法を用いて強化する必要があります。

障害復旧計画は、計画が実行される運用環境そのものと同じくらい欠かせない要素です。想定外のミスと大規模な障害はどちらも起きるものですから、備えあれば憂いなしです。

オープン ソース

マイクロソフトでは DevOps への取り組みと並行して、オープン ソース ソフトウェア (OSS) コミュニティにユーザーおよび投稿者として積極的に参加してきました。OSS のワークフローは、共有、再利用、コラボレーションが重要な要素です。OSS では、緩やかに結合された共同サービスをそれぞれ独立してリリースすることが推奨されています。

自社の問題に適した OSS ソリューションがあれば、それを使用します。再発明することはありません。そして、幅広い価値を備えたサービスを作成すれば、それを共有します。人気の高いコンポーネントを開発したエンジニアには報酬を提供し、使用が承認されたコンポーネントを社内のだれもが検索できるようにしています。どのエンジニアでも社内のあらゆる問題に改善策を提案できます。それと同時に、社内には企業としてのコード ガバナンスと長期にわたってサポートされる製品に関する要件があります。

インシデント発生から学びを得るまで

インシデントは、そこから何かを学び、それを伝達するまで終わりではありません。マイクロソフトでは「5 つの Why」[8] を用いて、複数の視点から次回に向けた改善を考察しています。お客様に影響を与えるサービスの停止が発生した場合は、標準の対応を実施します。問題の詳細とそこから学んだことを、対応担当エグゼクティブである Brian Harry がブログで説明します。ブログはそのまま公的な記録となり、私たちが改善を約束する言質にもなります。[9]

このブログに対するお客様の反応は注目に値します。ブログに記されているのは私たちの失敗がどれだけお粗末なものであったかを示す内容ですが、お客様からは感謝の言葉が寄せられています。この透明性が歓迎されているのです。なぜなら、通常のベンダーとお客様のやり取りよりも、はるかに有意義な関係がそこにあるからです。

ビジネスと開発をつなぐ

以前は、ビジネスと開発の利害関係を 1 つにまとめることは、その決定の影響を確認できるまでに長い時間がかかるため困難でした。DevOps によって、これが変わりました。マイクロソフトでは、Web 検索であっても試用版であっても、ユーザーが最初に接触してきた瞬間からユーザーのエクスペリエンスを向上させたいと考えています。テレメトリを通じて全体的なエンゲージメント統計を注視し、得意顧客が考えていることを理解するために 1 対 1 の働きかけを実践しています。

たとえば、お客様からのアプローチはじょうごの各段階を経ていくと捉えています。じょうごは複数あります。アプローチは、Web ページから始まり、試用版を経て、Azure やオンプレミスへのデプロイに至ります。じょうごのいずれかの段階で通過率が思わしくなければ、改善する方法について仮説を立て、加える変更を実験し、結果を測定します。このようなビジネス上の実験を、ライブ サイトや、既存のアクティブ ユーザーの月間成長率など、サービスの他の要素と共にレビューします。自己満足のメトリックではなく実用的なメトリックに焦点を当てるよう注意を払っています。

また、そのようなメトリックはお客様から寄せられる定性的なフィードバックと組み合わせることが最適であることも理解しました。マイクロソフトでは、さまざまな方法でフィードバックを集めています。その中で最も直接的なアプローチが「トップ カスタマー プログラム」です。このプログラムでは、エンジニアリング チームのメンバーを対象に、マイクロソフトのサービスを利用している上位ユーザー アカウントの 1 つに自ら進んで親密な関係を築き、月に 1 回程度は会話する機会を持つよう依頼しています。このコミュニケーションを通じて、良い点、悪い点、要望、わかりにくい部分など、数字では見えない情報を質問を通じて引き出すことができます。マイクロソフトには、「チャンプ」という社内サークルがあります。チャンプは主に MVP や親密なお客様で構成されており、私たちはほぼ毎週 Web ミーティングでチャンプと一緒にアイデアのプレビューを実施しています。ミーティングでは、仮説段階のシナリオを主にストーリーボードとして評価し、実行に移すはるか前の時点で、優先順位について意見を交換します。メンバーをつないでいるのは uservoice.visualstudio.com などのチャネルです。そこでは、だれもが提案や投票を行うことができます。

学びの共有: 効果的な DevOps のための 7 つの習慣

DevOps に移行する過程で、私たちは 7 つの取り組み領域で成長を評価しました。私たちは、この 7 つの領域を総合して「アジャイルの次の 10 年」と捉えています。

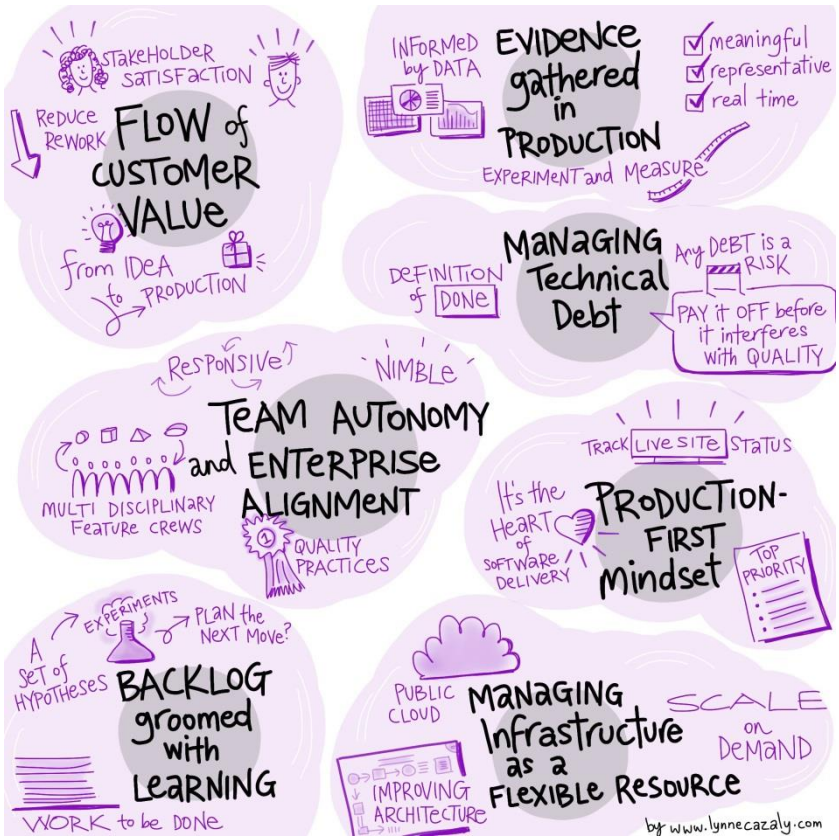


図17. マイクロソフトが DevOps の習得に取り組むにあたり、継続的に強化を図っている 7 つの取り組み領域

迅速なスケジュール調整と俊敏なチーム。これはアジャイル手法の場合と同じですが、さらに軽量です。複数の専門分野で構成されたフィーチャー クルーが、共通のプロダクト バックログから取得し、未完了の作業を最小限に抑えて、スプリントが終わるごとにライブ展開の準備が整った作業をリリースします。

技術的負債の管理。社内にある技術的負債はリスクになります。ライブ サイト インシデントなど予定外の作業が生じる原因となり、目標とするリリースに支障が出ます。私たちは負債項目には常に細心の注意を払い、提供するサービスの品質に悪影響を及ぼす前に清算できるようスケジュールを調整しています (前述した VS 2013 リリースの事例のように判断を誤ることもありますが、コミュニケーションでは常に透明性を心がけています)。

価値のフロー。価値のフローとは、バックログのランク付けを常にお客様にとっての重要事項に基づいて行い、お客様への価値の提供に重点を置くことを指します。アジャイルの最初の 10 年間、私たちは絶えずこのことを話してきましたが、現在は DevOps のテレメトリを使用して、成功率はどの程度か、軌道修正する必要があるかどうかを評価できます。

仮説ベースのバックログ。DevOps 以前は、プロダクト オーナーが関係者から得られた最良の入力値に基づいてバックログを調整していました。現在は、バックログを一連の仮説として扱っています。仮説は実験に発展させて、その仮説を支持または反証するデータを収集する必要があります。得られた証拠をもとに、バックログの次の動きを判断し、保持 (続行) または方向転換 (軌道修正) することができます。

証拠とデータ。マイクロソフトは、正常性、可用性、パフォーマンスなどのサービス品質の監視はもちろんのこと、使用状況を理解し、バックログの仮説に関する証拠を収集するためにも、あらゆる要素を利用しています。たとえば、ユーザー エクスペリエンスに対する変更を実験し、じょうごの変換率に対する影響を測定しています。また、平日の利用者と週末の利用者など、コホート間で使用状況のデータを比較し、それぞれのエクスペリエンスの改善方法について仮説を立てています。

運用優先の思考。せっかく収集したデータも、サービスの品質が一貫して高くなければ信頼することはできません。マイクロソフトでは常にライブ サイトの状態を追跡しています。ライブ サイト インシデントが発生すれば根本原因を解決し、パフォーマンスに外れ値があれば事前に識別して低下が発生している原因を特定します。

クラウドへの対応。サービスを 24 時間年中無休で提供するには、アーキテクチャを継続的に改良して独立性の高い個別のサービスにリファクタリングするか、柔軟性に優れたパブリック クラウドのインフラストラクチャを使用する必要があります。クラウド (マイクロソフトの場合は Azure) であれば、容量の追加が必要な場合にも取得できます。マイクロソフトでは、少数の計画的な例外を除き、新しい機能はすべてクラウド ファーストで開発してからオンプレミスに移行しています。このアプローチの採用によって、私たちは新機能をリリースするにあたり、持続的な使用から継続してフィードバックを受け取り、新機能を飛躍的に強化できたと自信を持って言うことができます。

後注

[1] 当時の詳細については、http://blogs.msdn.com/b/bharry/archive/2013/11/25/a-rough-patch.aspx (英語) をご覧ください。

[2] 具体的な例については、Gartner による ALM のマジック クアドラントをご覧ください。

[3] Eric Ries 著、『The Lean Startup』(2011 年)

[4] Jeffrey Liker 著、『The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer』(2003 年)

[5] http://aka.ms/redblueteam (英語) をご覧ください。

[6] https://msdn.microsoft.com/ja-jp/library/dn589784.aspx (英語)

[7] http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html (英語)、http://blog.smarx.com/posts/wazmonkey-chaos-monkey-for-windows-azure (英語)

[8] http://en.wikipedia.org/wiki/5_Whys (英語)

[9] 例: https://social.msdn.microsoft.com/Search/en-US?query=outage&beta=0&rn=Brian+Harry%26%2339%3bs+blog&rq=site:blogs.msdn.com/b/bharry/&ac=4 (英語)