大規模サーバのハードウェア会社

James Carpenter

このケーススタディの音声版は *こちらからご覧いただけます*

概要

長期的な成功を収めるには、プロダクトの境界（バウンダリー）を広く取り、プロダクトの全範囲を対象とするフィーチャーチームが必要だと強く信じています。コンポーネントチームから始めて、よりクロスファンクショナルなコンポーネントチームと、真のフィーチャーチームへと積極的に移行するための段階的なアプローチは、追求する価値があると信じています。

ナカシマのモジュラー・コンピューティング・システム部門におけるこの戦略の利点は以下の通りです。

- 拡張されたコンポーネント境界内での**適応性**と**価値の提供**をローカルに改善する
- **品質向上**を実現する技術的なプラクティスの改善と、さらに改善することで何ができるかの認識向上
- 拡張コンポーネントに関連する**不具合**の早期発見と解決
- 拡張コンポーネントチーム内の従業員の**コラボレーション、エンゲージメント、学習**の改善
- **組織的な障害**や、さらなる組織的な変革の必要性への認識が高まった

私たちの成功は、皆さんにとって刺激的であり、勉強になるものです。同様に、私たちの失敗を繰り返さず、新しい失敗をしながら学ぶことにエネルギーを使ってほしいと思います。

ざっと読むためのヒント

このケーススタディは非常に長く詳細です。さっと流し読みしたい方には、以下のみを読むことをお勧めします。

- 概要セクション
- すべての図とそのキャプション
- 結論セクション

目次

LeSS 適応の全体像

プロダクト概要組織をどう巻き込んだか

ナカシマのモジュラー・コンピュート・システム（MCS）部門は、ハードウェア、ソフトウェア、ネットワーキング・コンポーネントを統合した製品開発に注力しており、これらのコンポーネントが一括して完全なプレバンドル型社内データセンター・ソリューションとして機能するようになっています。MCS は、Amazon Web Services のようなプラットフォームを、フォークリフトでオンサイトに配備するようなものです。典型的なエンドユーザーは、大手銀行、保険会社、携帯電話プロバイダー、科学研究機関などである。

規模感としては、ナカシマの MCS 事業部だけで数千人規模になる。そのほとんどが、サンフランシスコ、ポートランド、ベンガルールといった大都市圏に集中している。

MCS 事業部の仕事は、すべて MCS の製品開発に集中している。基板製作、金属加工、組み立てなど、量産品の製造はすべて MCS 事業部が作成した仕様書をもとにアウトソーシングしている。

フィールドサポートは、ナカシマのもう一つの部門が担当している。フィールドサポート部門は、ナカシマの全製品に対してシームレスなカスタマーサポートを提供することに重点を置いています。電話機などのネットワーク製品、テレビ会議システム、ワイヤレスネットワークソリューションなど、MCS とは関係のない製品まで幅広くサポートを行っている。

実際、フィールドサポート部門には、MCS のサポートに特化したスペシャリストが存在する。このスペシャリストは、現場のお客様の実態や、MCS の大規模かつ多様なお客様の間でどのように変化しているかを観察するのに非常に有利な立場にあります。そのため、MCS 製品を開発するエンジニアは、これらのフィールドサポートスペシャリストと直接協力することによって、最も有用でフィルタリングされていないフィードバックを得ることができます。

MCS のサポート体制は、一般的なコンシューマー向け製品とは異なります。迅速な対応が可能な、大規模なテクニカルサポートです。MCS を開発するエンジニアと同じように、フィールドサポートのスペシャリストも、専門分野に精通し、高い報酬を得ている。

トレント・ガンベールは、MCS 事業部のプロジェクトマネジメントグループを率いてきました。このグループは、MCS の開発全体を調整する 10 人弱のプロジェクトマネージャーで構成されています。日々のプロジェクトマネジメントの多くは、MCS 事業部内の様々なエンジニアリングディレクターが担当していました。MCS 部門における私の最初のスポンサーシップは、トレントと、彼の副社長と当時の上級副社長兼 GM の積極的な関与によってもたらされたものです。トレントは、私が以前ナカシマの別の部門で仕事をしていたことを知っていました。

初期のアジャイル適応の焦点について

当初の目的は、ほんの一握りの既存の（表向きの）「アジャイル」チームを評価し、意味のある改善を支援し、特定できた部門の他の有望な領域でアジャイル適応を開始することでした。

数日間の調査の後、今やっている「アジャイル」な取り組みが、アジャイルではないことが明らかになりました。たとえば、次のようなことです。

- 経営陣は日常的に見積もりを「コミットメント」として扱っていた。

- 自動化されたユニットテストのようなアジャイルエンジニアリングプラクティスが実践されていなかった。
- スプリントレトロスペクティブのような「検査と適応」のためのイベントがなかった。
- チームは自己管理からかけ離れていた。

既存の、いわゆるアジャイルな取り組みを最初に調査した後、私は学んだことをトレントと話し合いました。私たちは、最も効果的にエネルギーを注ぐべき場所は、変革の成功に最も貢献すると考えられる分野に、機能横断的かつコンポーネント横断的な新しいスクラムチームを作ることであると判断しました。各チームは、開発、テスト、リリース、その他の活動のすべての側面を実行する能力とともに、関係するさまざまな技術のあらゆる側面に取り組むためのスキルと権限を持つ必要があります。

私が既存の取り組みを調査したところ、上級管理職の大半が、健全な自己管理型チームに対する真の理解や経験を持っていないことが明らかになりました。これでは、より持続可能な部門全体の組織改革に十分な関心を持たせることは難しいでしょう。トレントと私は、１つのスクラムチームがスターチームとして成功すれば、より大きな賛同を得られると期待していました。

しかし、「構造的な要因のいくつかに対処できなかったことが、結局、それまでの成功の大部分を台無しにしてしまった」のです。構造的な要因のうち、大きなものは次の２つです。

- 報告関係の再構築を適切に行わなかったため、一部の管理職がチームメンバーの業務から目を逸らしてしまった。
- 新しい機能は、ナカシマ全体としては非常に大きな節約になるにもかかわらず、経営陣のインセンティブに付随するものであった。

これらについては、【ラーマンの組織行動第一法則と第五法則の発現】の項で詳しく解説しています。

私たちはまず、MCS 製品にエンド・ツー・エンドの診断機能を追加することに焦点を当てた１つのスクラムチームを立ち上げました。エンドツーエンド診断のスターチームは、

このケーススタディの主要な焦点ではありませんが、初期の部分的な成功は、より広範な BIOS 関連の取り組みへの扉を開くのに役立ったのです。

エンドツーエンド診断の取り組みは、BIOS の取り組みで遭遇した課題のいくつかを明らかにし、予兆を示すのに役立つので、診断チームの取り組みについては、後でもう少し詳しく説明します。

BIOS グループを巻き込んだ LeSS 指向の採用

診断のスターチームの初期の成功と、私が MCS 部門の中で社交的であり続けたことが、BIOS グループの米国にいるエンジニアのディレクターである ミーチャ の興味を引き、最終的に加入させることになったのです。ミーチャ と私が率先して行った BIOS グループ内での LeSS 指向の適応が、このケーススタディの主な焦点となります。

BIOS ドメインは非常に複雑でした。加えて、BIOS 開発だけに限定されたコンポーネント境界がありました。それでも LeSS 指向の構造から大きな利益を受けるようなレベルでした。長年の習慣的な問題を解決するのに役立ったのです。

また、BIOS ドメインは、MCS システム全体のフルフィーチャー化に向けた自然な進化の道筋を提供するものでした。インテルが CPU を生産開始する前に、生産前の新世代のインテル CPU を MCS 製品に迅速に吸収する能力は、市場の関連性を維持するために重要でした。これは、すべての必要な BIOS 機能がプリプロダクション CPU 上で正しく動作することを確認し、同様にブレードアーキテクチャの改訂と拡張が CPU 統合の観点から適切に動作していることを確認することを含みます。さらに、MCS 管理ユーザーインターフェイスで公開されるすべての CPU 管理サービスが動作していること、およびインテルが量産前の CPU で導入する新機能をサポートするために拡張されていることも確認する必要があります。

上記のすべては、MCS 製品全体に対する自然な LeSS Huge リクワイアメントエリアと見なすことができます。

LeSS フレームワークのルールでは、LeSS-Huge 採用の各リクワイアメント領域は、「最初に」完全な LeSS 構造を確立する必要があるとされています。ミーチャと私は、BIOS グループの米国を拠点とするメンバーを最初から完全に再編成しましたが、上級管理職との連携や賛同があれば、より充実した、より適切な製品のフィーチャーチームから始めることができたはずです。

私たちは、できる限りのことから始め、徐々に BIOS コンポーネントの境界を、さまざまなコンポーネント層を通して上方に広げていきました。「フィーチャーチーム Adoption Map for BIOS」と「Initial BIOS Component Boundary」、「Expanded BIOS Multi-Component Boundary」図は、この段階的な拡大を示しています。このアプローチは、「Large-Scale Scrum」で説明されている「フィーチャーチーム Adoption Map」ガイドと一致していることに、賢明な読者はお気づきでしょう。More with LeSS_で説明

されている_フィーチャーチーム Adoption Map_ガイドと一致していることにお気づきでしょう。

BIOS の LeSS 指向の適応活動は、ちょうどミーチャと私が活動を始めた頃に起こった上級副社長/GM の交代によって妨げられました。もし、当初のシニア・バイスプレジデント/GM と同じレベルの経営陣のサポートがあったなら、もっと幅広く、持続的な成功が達成されていたことでしょう。

私は、ナカシマの MCS 部門における LeSS 指向の採用活動の成功例と失敗例の両方から、多くのことを学びました。このケーススタディが、私たちの成功例から、そして失敗例からさらに多くのことを学び、皆さんの旅立ちの指針となることを願っています。

ハードウェア開発ではなくファームウェア開発に焦点をあてる

これまで、リリーススケジュールの予測に関する問題の大部分は、MCS 製品開発グループのファームウェア（BIOS、ネットワーク・インターフェイス・コントローラ、シャーシ・コントローラなど）および上位のソフトウェア（MCS Administrator、診断など）開発面で発生していました。ハードウェアの開発面は、はるかに予測しやすい傾向がありました。

実際には、外部製造業者の製造仕様設計に携わる人たちとの会話はわずかな時間しかありませんでした。ハードウェアの設計に携わる技術者に LeSS の採用を拡大することは、長期的には意味のあることですが、当初の焦点には入っていませんでした。

実際、私が一緒に仕事をした診断チームも BIOS チームも、ハードウェアエンジニアと積極的にコラボレーションする必要性はあまりなかったのです。ファームウェアの開発者が必要なときはいつでも、ハードウェアの設計エンジニアと話をすることができましたが、それは頻繁に起こることではありませんでした。私が MCS 部門にいた頃は、MCS は比較的成熟した製品でしたので、ハードウェア設計の変更は小規模で漸進的なものになる傾向がありました。将来的には、ハードウェアとファームウェアの開発をより緊密に統合し、部門横断的なチームによるイノベーションを促進できる可能性があります。

Actual MCS Product Boundary

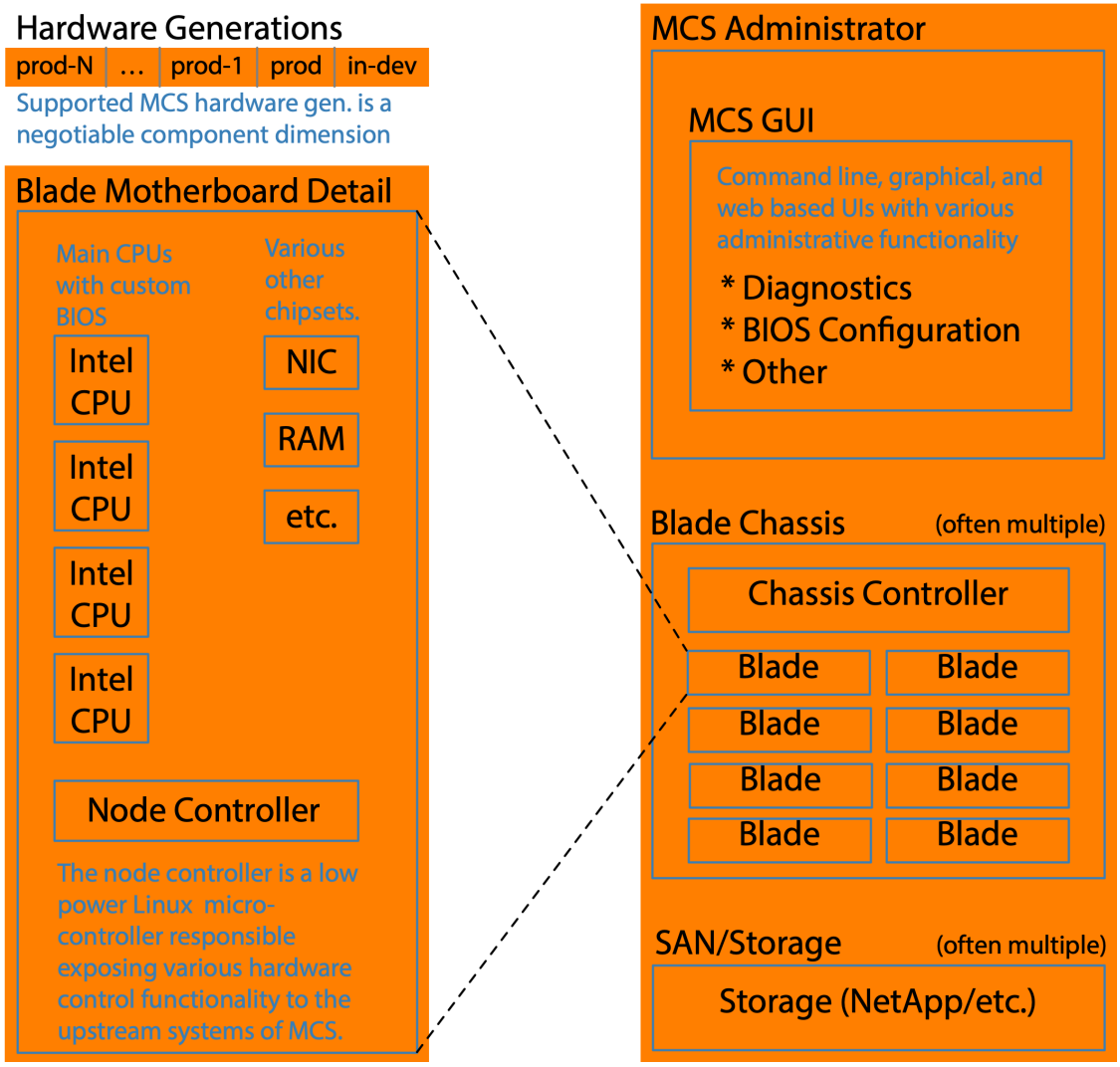

図１：「製品管理グループ、ナカシマ事業部の境界、および外部顧客の観点から、自然な製品境界は、ネットワーク、計算、およびストレージ機能のシステム全体を含んでいます。

他のシステム間の結合は十分に標準化されており、さまざまなベンダーのデータセンターのハードウェアやソフトウェアと互換性があるため、さらに広い製品境界を設けることは可能ですが、それほど現実的ではありません。これらのシステムは、より大規模なテストスコープにおいて重要な役割を果たしますが、モジュラー・コンピュート・システム・テストの焦点ではありません。

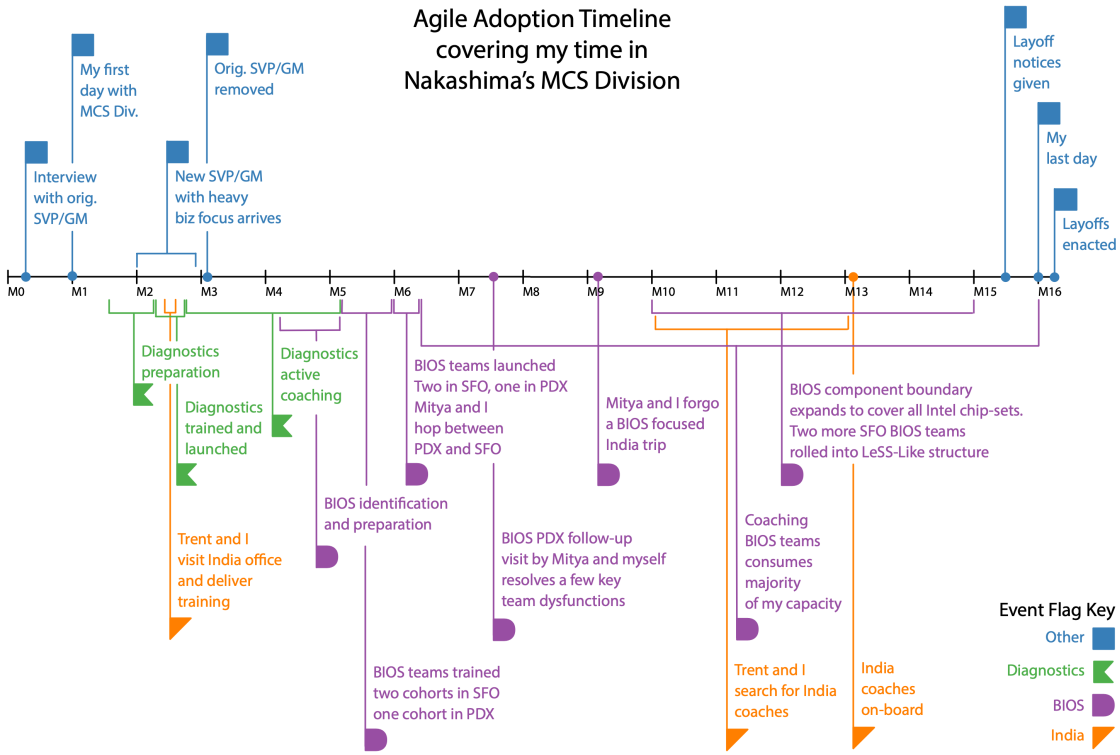

図 2: さまざまな LeSS 適応活動や経営陣の交代が重なると、全体のストーリーアーチがわからなくなりがちです。このタイムラインが追跡の助けになることを願っています。

スクラムチームのメリットをアピールする

MCS 部門では、当初、ハードウェアとファームウェアの開発作業においてスクラムが意味を持つかどうかにつ いて、非常に大きな懐疑論があった。ファームウェアとハードウェアの取り組みは、典型的なエンタープライズソフトウェア開発とはどこか異なっており、したがってスクラムとの相性は良くないと考えられていました（1980 年代のハーバード大学によるスクラムのルーツ研究は、ハードウェア製品を対象としていたので、皮肉なことです）。この視点は、MCS 内の既存の「アジャイル」開発の取り組みがすべて「なんちゃってアジャイルだった」という事実によって強化されていました。

この文脈から、有用な結果を実証できる健全なスクラムチームを立ち上げるための有意義な多成分の境界を特定するために、フィーチャーチームアダプションマップを使用して段階的な LeSS 採用ガイドを適用することは理にかなっていたのです。つまり、問題を解決するためのすべてのスキルを持った小さなチームが、問題を抱えた顧客と直接仕事をし、このチームが顧客の問題を反復的、適応的、漸進的に解決することの価値を実証することに意味があったということです。

私たちは、それを実証することで、さらなる「組織的な後ろ盾」を得ることを望んでいました。

- ファームウェア/ハードウェアを使ったスクラム
- 自己管理するチーム
- 顧客との協力関係の改善
- 改善された価値提供
- 文化に対する構造の影響
- コード品質の向上
- 契約ゲームの回避

パイロット版マルチコンポーネントの望ましい特性

上記のデモンストレーションの目的を達成するために、私たちは、パイロットスクラムチームが集中するための MCS 製品の適切な機能セットを特定する必要がありました。私は、製品全体の自然な LeSS Huge リクワイアメントエリア（RA）に著しくマッピングされる、大きく分離された複数のコンポーネントのセットを特定できれば、成功の可能性が最も高くなると認識していました。（RA はコンポーネントやアーキテクチャの境界によって定義されるものではありませんが、機能的な RA と関連するコード群が大きく重なることがあり、この初期段階のケースでは、採用を単純化する上で有利であったことに留意してください）。

以下の基準は、私たちが特定しようとしていた複数のコンポーネントのグループを定義するものである。私たちは、次のようなコンポーネントのグループを求めました。

- MCS の「前面から背面へ（よりユーザに近いところからシステムのバックエンドへ）」通過するもの。
- 単一のチームの能力に適切である。
- 顧客に有意義な価値を提供する
- 成功したデリバリーは政治的・組織的に無視できなくなる
- チームがリリースの圧力から保護される可能性があるもの
- ウォーターフォール開発者の軍団による悪習を避けるために、少なくとも適度に MCS システムの残りの部分から切り離されたコードベースがあること。

もう一つの基準として、私たちはこの基準なしでも生き残ることができましたが、次の基準も見つけることを希望していました。

- ウォーターフォールの MCS のリリースとは別にリリースできるコンポーネント群、それによって実際の顧客からのより迅速なフィードバックが得られること

パイロットスクラムチームのための適切な診断機能セットの特定

トレントや社内の他の人たちと熟考を重ね、良い候補のコンポーネントを見つけることができたのです。結論は？MCS の診断機能セットは、リストのすべての項目を満たすため、非常に良い選択であることが判明しました。

顧客サポートコストの大幅な節約とそれに伴う評判の向上は、政治的な援護射撃になると同時に、現場サポート部門からの熱狂的な支持を得ることができました。診断機能は、私たちの基準を満たす唯一の機能セットでしたが、その適合性は素晴らしいものでした。

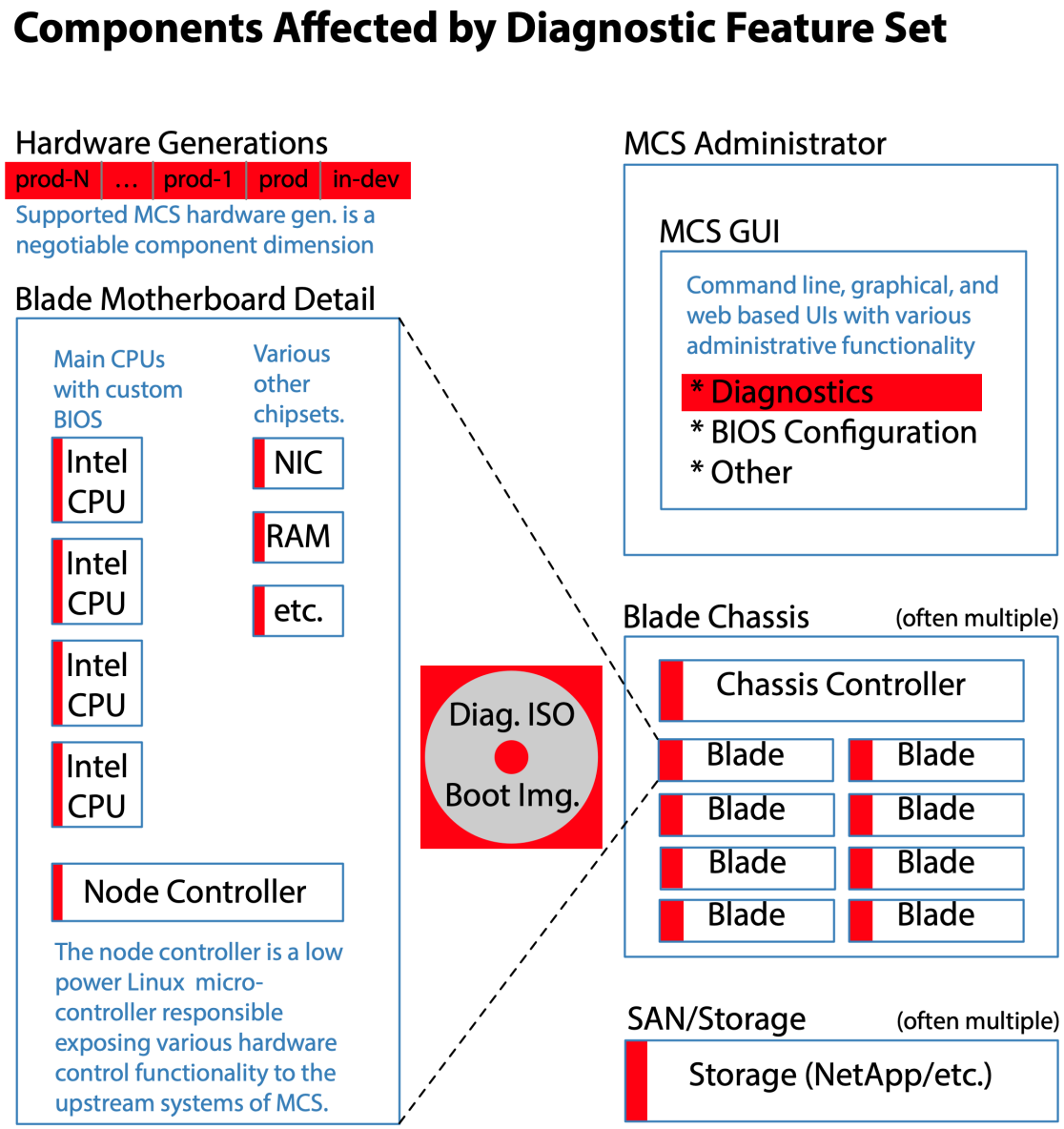

図 3: 診断コードの大部分はカスタム ISO イメージと MCSA の一部の診断に焦点を当てたコードにカプセル化されていますが、診断機能は広く焦点を当て、すべてのハードウェア世代に関連するものです。MCS システム全体の各コンポーネントの詳細な知識がしばしば必要とされる。ある MCS コンポーネントのファームウェアにプローブ機能がない場合、診断チームはそれを追加する。

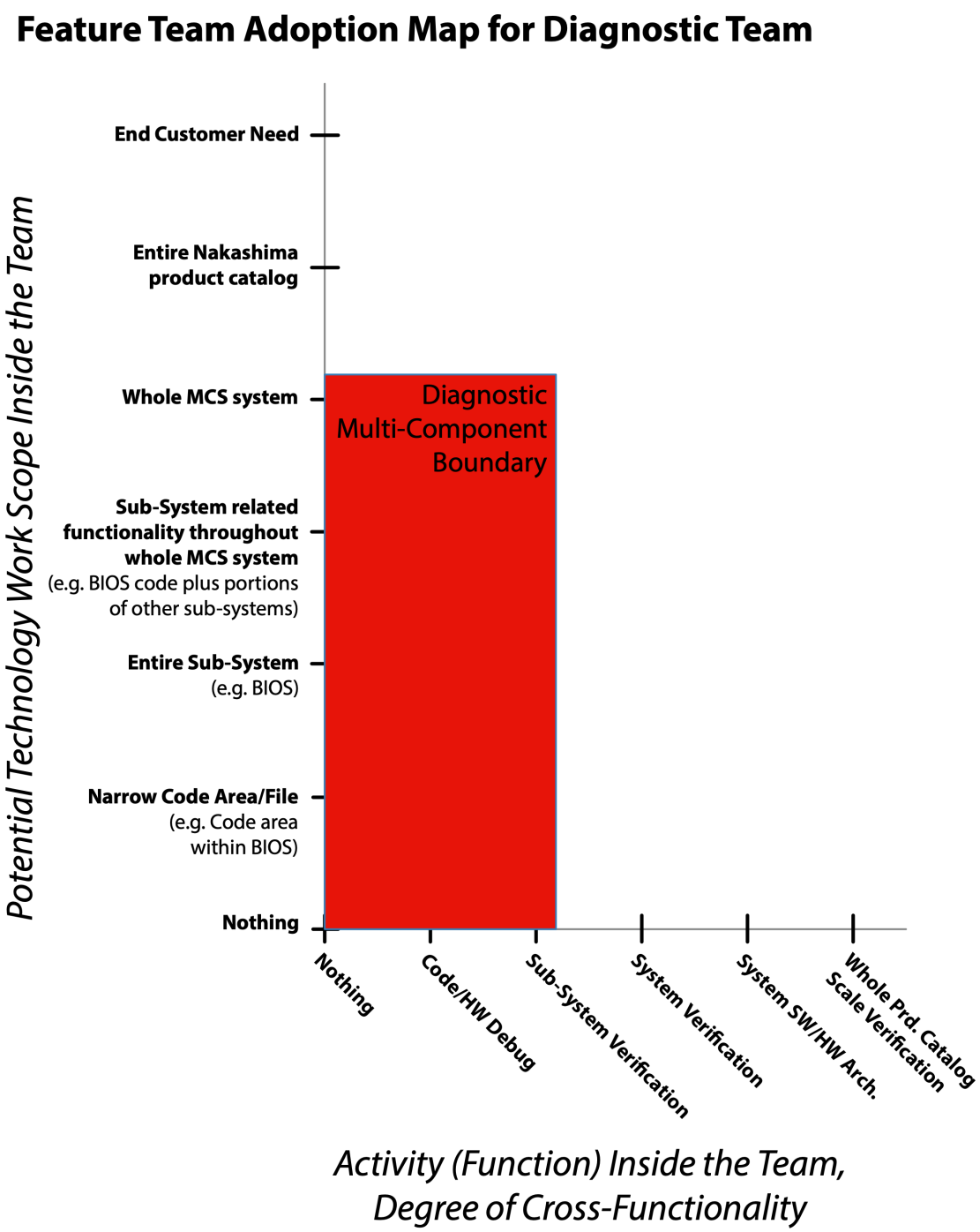

図 4: 診断機能は、顧客がオンサイトの MCS システム、特にハードウェアコンポーネントの故障の問題を診断するのを支援することに焦点を絞ったものである。しかし、焦点は狭いですが、スコープは全体の MCS システムにまたがっている。

*フィーチャーチームアダプションマップの使用に関する詳細情報は、以下を参照してください。*https://less.works/less/adoption/feature-team-adoption_map

診断チーム行動実績

最初の立ち上げから、人々は以前とは違う行動をとるようになりました。4〜5 回のスプリントの後、変化した行動には次のようなものがありました。 * 診断スクラムチーム全体とフィールドサポート部門のエンジニアとの積極的な協力関係 * 個人としてではなく、共同で作業するチーム、Sprint ごとに段階的に調整と改善を行うチーム、常に出荷可能なコンポーネントを生産するチーム、主要な利害関係者と豊かで直接的な協力関係を確立するチーム。 * フィールドサポートと別の協力部門が作成した指標に大きく影響されたマルチコンポーネントバックログの発注。これらの指標により、私たちが概念化したほとんどの診断能力について、月々の推定遅延コストを割り出すことが可能になりました。優先順位の高い診断機能で予測される節約額は、驚異的なものでした!

診断チームが技術的に達成したこと

明確かつ拡張された「Done の定義」、効果的なスクラムイベント、少量の技術コーチング、そして契約ゲームの回避が組み合わさり、技術的な実践に大きな違いを生み出したのです。

4〜5 回のスプリントの間に、以下のことが容易に観察できるようになりました。

- 診断スクラムチームが触れた MCS の C++と Python のコードのあらゆる部分において、ユニットテストが自動化された。
- 自動化されたユニットテストは、診断に特化したコードと MCS コードベース全体の一部に対して、より良く作られた、より問題の少ないコードのための強制機能として作用していました。
- MCS システムのために、以前より改善された、より広範な自動化された統合テストの積極的な開発。

診断チームの立ち上げのステップ

成功には何の不思議もなかった。手順は以下の通りです。

- チームをサポートし導くことに貢献する人格を特定するために、LeSS ガイド **Temp_orary_Fake Product Owner** を診断用マルチコンポーネントに適用し、彼の決定が様々な利害関係者によって尊重されるために十分な地位と政治的影響力を持つ人物としました。
- この人物は、社内の他者から指示された要件をこなし、マイルストーンに準拠する「契約ゲーム」をまだ行わなければならないため、「偽の」プロダクトオーナーとなりました。
- 必要な様々な MCS サブシステムを横断して作業するために必要な（または学んだ）すべてのソフトウェアとテストエンジニアリングのスキルを持つ 8 人のメンバーの開発チームを組み立てました。

- 開発チームの共同作業スペースとして使用するために、中規模の会議室をフルタイムで所有した。
- スクラムチームの全メンバーと主要なステークホルダーのために、数日間の正式なクラスルームトレーニングを提供しました。
- 数日間に渡る専用ミーティングと、様々な事前ミーティングやフォローアップミーティングから成る、共同憲章作成作業を実施。これにより、ステークホルダーとスクラムチームメンバーとの間で、全体的な整合性をとることができました。
- 私は、最初の数スプリントの間、経営陣、プロセス、技術的なコーチングを積極的に行い、その後、MCS 部門の BIOS グループにおける LeSS 指向の導入に力を注ぎました。

大規模スクラムの「はじめに」ガイドを参照してください。チームの立ち上げに関する追加のガイダンスについては、『大規模スクラム：LeSS でさらに進化する』の _Getting Starting_ガイドを参照してください。LeSS 構造の中で複数のチームを立ち上げるという観点から書かれていますが、1 つのパイロットスクラムチームを立ち上げる場合にも同じように適用できます。

診断チームの写真とアーティファクト

図 5：診断スクラム開発チームの写真です。ペアリングとスワーミングは時間とともに一般的になったが、フルモブプログラミングは全く普及しなかった。このチームは、各スプリントの終わりに出荷可能なインクリメントを提供するために必要な、テストと開発の両方の才能を持っていた。開発チームは、物理的なタスクボードを使用していました。私たちが引き継いだ会議室は、私たちが望んでいたよりも少し小さかった。

Second Definition of Done (Firmware Feature Team in Waterfall Ecosystem)

Version: 2.03

Date: July 5, 2017

Common

- ❑ Zero bugs at the end of the Sprint within the Dev team's control to fix.
- ❑ Any similar functionality between the stand-alone and integrated solution has been factored into common shared libraries.
- ❑ Manually validated against each available hardware platform profile.
- ❑ Release notes updated and published to common location.
- ❑ Configuration information and Release note information saved in source control in a manner which makes it easy for the technical writing team to write documentation for the release.
- ❑ Peer Review completed. (Pair programming counts as peer review.)
- ❑ Demo to Product Owner

Specific to Stand-alone Solution

- ❑ Automated unit test created for both positive and negative cases for any new or modified functionality within custom code. Not required for third-party libraries developed outside the team.
- ❑ Manual Regression executed and passed for all platforms.
- ❑ Passed all solution level test cases for new defects (manual testing today).
- ❑ Final release image build by centralized build systems team is successful.
- ❑ Optimal image size should not increase beyond 100MB.

Specific to Larger Solution

- ❑ Code committed to main branch of overall solution.
- ❑ Automated smoke tests extended to cover any new or modified functionality.
- ❑ Existing commit tests pass.
- ❑ No new static analysis errors introduced.
- ❑ Manual feature test passed.
- ❑ No relevant open defects.
- ❑ No relevant open regression defects.

図 6：診断チームが使用した「Done の定義」は、数スプリントの後、ここに示すように進化しました。インクリメントのスタンドアロン部分は、各スプリントの終わりまでにフィールドサービス部門に提供されました。MCS の統合診断機能を最終顧客に提供するには、ウォーターフォールで開発された MCS 製品のリリースを待つ必要があった。この例の Definition of Done は、その他の文脈とともに、 Forging Change の表 9.2 に記載されています。

診断チーム文化的関連要素

試験的な取り組みに関するいくつかの文化的観察を以下に列挙する。これらの文化的観察の多くは、BIOS グループにおけるその後の LeSS 指向の採用のより良い理解にも関連するものである。

ラーマンの組織行動第一法則と第五法則の発現

ラーマンの組織行動の法則の第一法則と第五法則は、以下の通りです。

- **組織は、暗黙のうちに、現状維持の中間・第一レベルのマネージャーと「専門家」の地位・権力構造を変更しないように最適化されている。**
- **（大きく確立された組織で）文化は構造に従う。そして、小さな若い組織・会社では、構造は文化に従う。**

ラーマンの組織行動の法則の全 5 項目は、LeSS のウェブサイトの構造のページ *https://less.works/less/structure/index* で見ることができます。

小規模な診断パイロットスクラムチーム内でも、第一法則と第五法則が明確に表れていた。効果的な構造とその結果としての文化におけるポジティブな変化は、診断チームと、診断チームが積極的に協力するステークホルダーに限られたものであった。一方、中間管理職層の大部分では、現状がほとんど変化していなかった。

診断チームが拡大した構成要素の境界は、MCS システム内の多くの異なる構成要素にまたがるものであった。これらのコンポーネントには、通常、エンジニアと地域ごとのマネージャーがおり、2 人の副社長のうちの 1 人にロールアップされていた。診断開発チームは、各マネージャーが関連するコンポーネントの知識を持つソフトウェアエンジニアやテストエンジニアを完全に配置することで結成されました。ウォーターフォール開発における納期のプレッシャーを考えると、マネージャー達は診断チームに人を割り当てることにとても抵抗がありました。最終的には、シニアバイスプレジデントの意向でそうすることになりました。

スクラムチームが稼動してからは、政治的な圧力とシニアバイスプレジデントからの正式な指示により、ファーストレベルとセカンドレベルのマネージャーは、診断開発チームの個々のメンバーに関係のない仕事をさせることをやめました。開発チームのメンバーの中には、いくつかの問題を解決するために、最終的に診断作業に関連するマネージャーの下に正式に配属されなければならない者もいました。

Larman の組織行動の法則が予言するように、スクラム開発チームメンバーが、MCS 製品全体からの納期プレッシャーに同時にさらされるエンジニアリングマネージャーに報告することは、非常に問題であった。副社長が協力することでこの問題は軽減されたが、組織図が完全にフラット化されることはなかった。開発チームのメンバーの多くは、この変更を完全に受け入れることなく、副社長の 1 人を通じて報告を続けていました。唯一の変更は、診断スクラム開発チームのメンバーの一部を、どのディレクターが担当するかということでした。

LeSS ブックガイダンスとの対照表

一見したところ、この診断パイロットは、LeSS の 3 冊目の本である「大規模スクラム」の中の「並列組織」ガイドと一致している。しかし、残念なことに、少なくとも 2 つの重大な相違点がある。

Larman と Vodde は、並列組織戦略について説明する際に、いくつかの注意点を挙げています。最初に挙げられている注意点は並列組織はお試しではなく、組織の報告系統を従来の組織とは別にする必要があります。

診断チームの有効なチームレベルの構造を抜本的に変更し、スクラムに合わせたが、正式な組織の報告関係は残ったままであった。この失敗により、チームは経営層の変化に対して過度に脆弱になった。

診断チームは契約ゲームの対象外であったにもかかわらず、診断チームのメンバーは、ウォーターフォールの取り組み全体の契約ゲームに従事していた中間管理職を通じて、報告を行った。中間管理職は、ウォーターフォールのリリースを控え、より多くの開発能力を求めるようになると、個々の診断チームメンバーの努力を方向転換させるという、以前のような行動を取るようになりました。当初のエンジニア SVP/GM は、このような機能不全から診断チームを積極的に守っていましたが、彼が去るとすぐに、診断チームの初期の成功を可能にした支援的な状況が徐々に侵食され始めました。

第二の違いは、診断チームが、いずれ並列組織を形成することになる多くのチームの第一陣としてではなく、スターチームとして捉えられていたことである。言い換えれば、この診断チームは、MCS のすべての開発を LeSS 指向の構造にゆっくりと移行させるという上級管理職によるコミットメントされた集団的決定を実行するための最初のステップとは見なされていなかったのです。

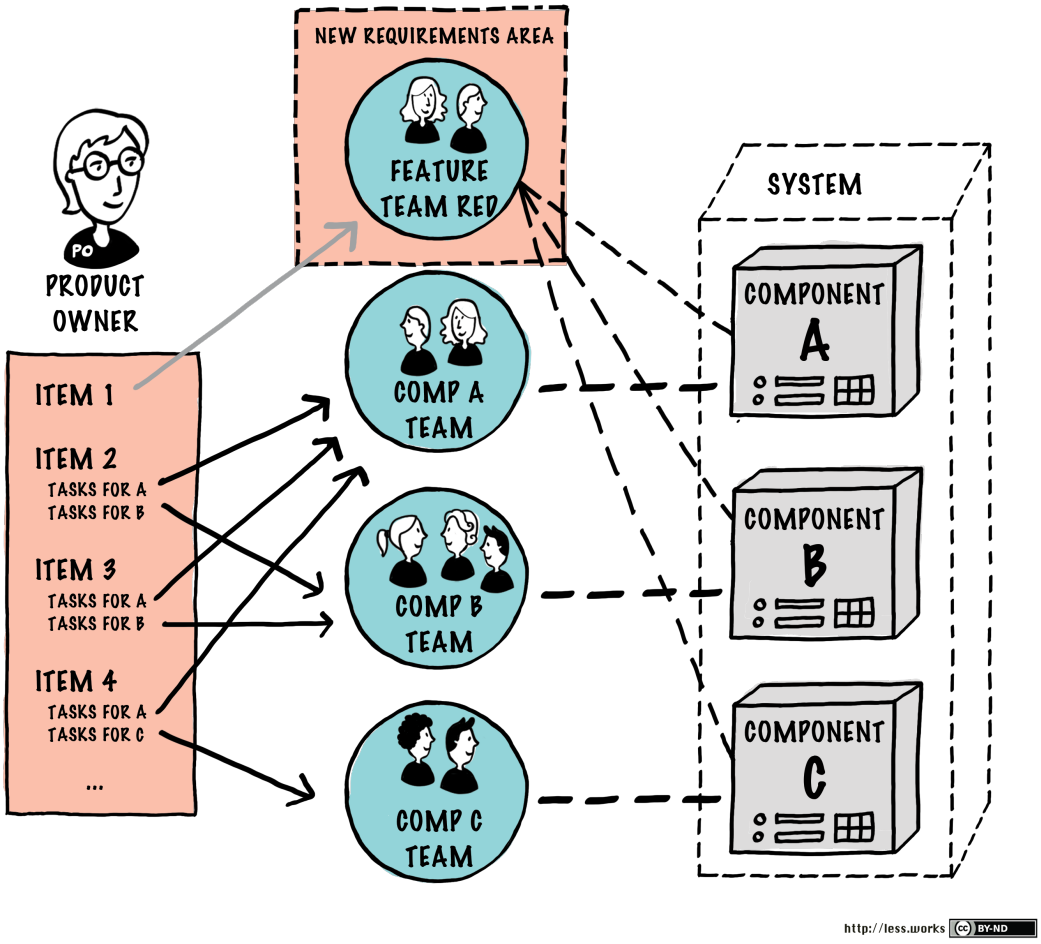

図 7: この図は、「大規模スクラム」の図 4.11 として掲載されています。More with LeSS の中で、フィーチャーチームへの移行ガイドの一部として、図 4.11 としてこの図が掲載されています。新たに形成された要件エリアにいるフィーチャーチーム Red は、すべてのコンポーネントチームと同じプロダクトバックログから消費していることがわかるでしょう。これは診断チームの状況とほぼ同じですが、日々の現実は少し違っていました。 診断チームは、契約ゲームの直接的な悪影響から解放された、真の自己管理チームであった。これとは対照的に、他の MCS チームのほとんどは、ウォーターフォール納品という状況の中で、コントラクトゲームにさらされていたのです。

診断チームの組織的な報告関係は、ウォーターフォールチームで行われていたコントラクトゲームのレベルより上の組織図と交差するように変更されることはなかった。この失敗は、最終的に、診断チームが成功し続けるために必要な支援的な状況を侵食する結果となった。この失敗は、当初のエンジニアリング担当の SVP/GM が去った後、ますます明白になった。

関連文献

LeSS 規則の最新版は、LeSS ウェブサイトに掲載されています。
https://less.works/less/rules/index

以下の LeSS ルールは特に関連性が高いと思われます。

- “LeSS のルール。プロダクトグループについては、LeSS の完全な構造を「最初に」確立すること。”

1 つのチームしか設立されていないため、明らかに達成できていません。診断チームレベルの構造は最初からできていたのですが、診断チームに関連する報告関係の変更と組織図のフラット化ができていなかったのです。

大規模スクラム。More with Less では、以下の関連ガイドを提供しています。

- *ガイド チームベースの組織を構築する*。このガイドでは、動的なマトリックス構造よりも安定した組織を持つことに特に重点を置いています。
- *ガイド LeSS の組織構造*。以下の引用が含まれています。“LeSS の組織にはマトリックス構造はなく、点線の管理者は存在しない”
- *ガイド フィーチャーチーム s への移行:* 様々な移行戦略のハイレベルな概観を提供します。
- *ガイド 一度に 1 つの要件領域*。LeSS の大規模な導入のための段階的なアプローチについて説明しています。
- *ガイド: 並列組織*: 採用戦略として、別組織を作ることについて詳しく説明しています。特に、記載されている注意点は洞察に富んでいます。
- *ガイド: フィーチャー・チーム採用マップ*: 図 4 で示されるように、フィーチャーチームの責任が徐々に拡大する様子を可視化するための、より深い洞察を提供します。
- *ガイド: フィーチャーチームの採用マップ Done の定義を進化させる_*: Done の定義を使用して、どのように漸進的な改善と様々な利害関係者との関連性をより明確に可視化できるかを説明します。
- *ガイド 始め方*: 個々のチームの立ち上げにも、複数のチームの立ち上げにも、同じように適用できます。

エグゼクティブレイヤーの交代を振り返って

エグゼクティブレイヤーの変更に関する考察。新しい SVP は、彼なりに非常に優秀で、技術的にもそこそこ精通していましたが、後任の短期間のチーフエンジニアスタイルの SVP のような技術的な見識に欠けていたのです。

SVP の下にいる既存の上級管理職は、優れたソフトウェアエンジニアリングの実践を行う適切なアジャイルチームのようなものに実際に触れる機会がありませんでした。そのような経験を積んでいない限り、さらなる変革への支持を得ることは困難だった。

契約ゲームの回避に失敗し、現状維持の権力構造を自己防衛しようとしたため、多くの人が時間をかけて行われた改善を評価し、受け入れることは困難だった。これは、前述した「ラーマンの組織行動の法則」の第一法則が予言する通りである。

もし、当時の私に知恵と技術があり、「Certified LeSS for Executives」コースのような、シニアマネジメントが変化を借りるのではなく、自分のものにすることに焦点を当てたエグゼクティブワークショップを実施する必要性を認識していたら、事態は違っていたかもしれないのです。 もし、私がまだ元の SVP のサポートを受けていた時にこの方法をとっていたら、元の SVP は、部門レイヤーの上に作用する政治的な力を乗り越えて、十分に思い切った変化を起こしていると見なされたかもしれません。

何はともあれ、インフォームド・コンセントのワークショップと組み合わせたエグゼクティブ・ワークショップは、ソフトウェア VP による受動的攻撃的な行動を浮き彫りにするのに役立ったかもしれません。

持続可能な文化の変革に必要な組織の構造的変化の大きさをより強く認識させることは、せっかく実現した漸進的な成果をあげる前に、改善の努力を止めてしまう危険性がある。しかし、このような経営者向けワークショップの後、元の SVP が LeSS 指向の採用に積極的であったなら、採用活動を覆すことはさらに困難だったでしょう。

新しい SVP が現れた時には、大規模なレイオフと多くの有意義な改善の破壊につながる賽は既に投げられていたのです。私はまだそれを知らなかっただけで、あと 1 年は知らないでしょう。

しかし、このとき達成された成果には、大きな意味がありました。パイロット診断チームが導入した診断機能は、数百万ドルのコスト削減を実現し続けています。レガシーの組織構造では、同等の品質の診断機能を実現することは不可能だったでしょう。BIOS の生産コード、BIOS テストインフラ、BIOS チーム内の組織的規範の多くの改善によるさらに大きな好影響は、完全に元に戻すにはあまりに大きな勢いを得ました。

啓蒙された人々は、二度と同じ世界を見ることはない。同様に、量産製品のコードに組み込まれた改良も、何十年も続く傾向がある。

BIOS 管理への関心

診断スクラムチームが軌道に乗り始めると、私は MCS の分野で別の分野に注力することを模索し始めました。私は再びトレントの組織力学の知識を活用しました。前回同様、経営陣のサポートが十分に得られる分野に焦点を当てながら、徐々にフィーチャーチームに移行していこうと考えていました。スクラムチームの診断の効果を考えると、もう少し広い範囲での取り組みが可能になってきた。とはいえ、コーチはまだ私一人でしたし、当時はコーチング能力を追加するための十分な資金もありませんでした。

ミーチャというディレクターがいたのですが、彼は当初はやや懐疑的だったものの、どんなメリットがあるのかに興味を示してくれました。数週間かけて、私はミーチャの考

えをさらに深めることができました。その際、ミーチャの仲間であるクリシュナ・ミシュラという人物が協力してくれました。クリシュナは、診断パイロット・スクラム・チームの活動で重要な役割を果たしました。また、クリシュナは、ミーチャのグループと頻繁に交流しているエンジニアのグループを管理していました。

ミーチャのグループは、MCS 用にカスタマイズされた BIOS を提供する役割を担っていた。BIOS は、起動時に計算ノード（ブレード）のハードウェアを初期化するためのファームウェアです。また、MCS 内のコンピュート・ノードで稼働しているどの OS に対しても、ランタイムサービスを提供します。MCS や競合製品の場合、カスタム BIOS は、ハードウェアに物理的にアクセスすることなく、各ノードのハードウェア設定をリモートで管理することを可能にする一部である。ハードウェアが遠く離れた、人手の少ないサーバーファームにあることを考えると、このアクセスは特に重要です。

BIOS 概要

カスタム BIOS コードは、MCS システム全体の中で最も困難で専門的なソフトウェア開発の側面であったと言えるでしょう。残りのシステムの多くは、成熟したチップセット上で動作する C++、Java、Python ミドルウェアで構成され、専用のアプライアンスハードウェア上でストリップダウンした Linux オペレーティングシステム内で実行されました。これらのシステムの中には、低消費電力やメモリの制約を受けるものもありましたが、BIOS エンジニアが対処しなければならないことに比べれば、大したことではありません。それにもかかわらず、経営陣が最も改革・改良に意欲的だったのが BIOS グループだったのです。

ナカシマでは、インテル社のプロセッサーのリリースに合わせて、カスタム BIOS のリリースを迅速に行うことが、市場からの要求に応えることでした。AMI が BIOS ファームウェアのベースを提供していたとしても、歴史的にエンジニアリングのやり方が悪く、部門横断的なマルチラーニングチームがなかったため、インテルの最新情報を得るためには、40 人以上の高度に専門化したハードウェア、ソフトウェア、テストエンジニアがカスタム MCS BIOS に取り組む必要がありました。

BIOS のメジャーリリースは MCS の全体的なリリースに対応する傾向がありましたが、より小さな独立した BIOS のリリースも時折行われます。独立した小規模なリリースは、通常、インテルハードウェアのマイナーリビジョンや、カスタム MCS BIOS のベースとなる AMI コードベースのマイナーアップデートによって動機づけられています。

BIOS 管理レポートライン内の変更に対する熱意と、ソフトウェア VP からの不十分なサポートを考慮すると、賢明なことは、ハードウェア VP と品質保証 VP の影響力の範囲内で、可能性の芸術を追及することでした。私たちは、カスタム BIOS だけに限定してコンポーネントの境界を定義し、BIOS チームが成熟するにつれて徐々に境界を広げていくことにしました。

マクロレベルではBIOSはMCSシステム全体の1コンポーネントに過ぎないが、ミクロレベルではLeSS指向の構造のあらゆる部分が完全に関連性を保っている。BIOSのコンポーネント自体は、このドキュメントの範囲外である非常に大きなサブコンポーネントのセットで構成されています。

MCS部門には、数千人のエンジニアがおり、全員がMCS製品の様々な側面に集中しています。BIOSコンポーネントチームの範囲内でエンジニアリングプラクティスとクロストレーニングを改善する前に、BIOSマルチコンポーネントの境界を拡大することから始めるのは問題があったでしょう。MCS全体のエンジニアリング手法は、長年のコントラクトゲームから予想されるように、比較的ひどいものであった。最初のBIOSコンポーネントの境界内には、品質の問題とエンジニアリングのサイロ化が十分すぎるほど存在し、私たちが提供できるコーチングの焦点をすべて消費していました。

LeSSでは、*https://less.works/less/adoption/coaching*で説明されているように、3つのレベル（組織、チーム、技術）でコーチングを行うようアドバイスしており、私はその3つをすべてカバーしていました。私はTrentがベンガルールで何人かのコーチを面接し採用するのを手伝いましたが、MCS部門が西半球で持つ唯一のアジャイルコーチは私だったのです。もし、私が在籍していた最初の2ヶ月半の間に、最初のエンジニアリングSVP/GMを失っていなければ、コーチング能力の状況はおそらく違っていたでしょう。新しいエンジニアリングSVP/GMは部門の縮小を考えており、アジャイル採用の取り組みを十分に理解していなかったので、私たちは幸運にもコーチング能力を持つことができました。

BIOS ボトムアップからの拡張

LeSSでは以下のように説明されています。

> LeSS Hugeの採用には、おおよそ2つのアプローチがあります。
>
> 一度に1つの要件領域を採用する
>
> チームの作業範囲、Doneの定義、プロダクトの定義を徐々に拡大する。
>
> – *https://less.works/less/less-huge/huge-adoption*

私たちは、1つ目のアプローチではありますが、2つ目のアプローチに従いました。当時は、1つの要件エリアが、よりインパクトのある組織改革を意味したのでしょう。

BIOSのLeSS指向の採用は、製品全体に焦点を当てた顧客中心主義という原則に直接起因するものではありませんでした。むしろ、LeSS指向の構造の仕組みは、BIOSチームが直面していた透明性、調整、品質の問題の多くを解決するためのソリューションと見なされた。

BIOSのマルチコンポーネント境界を拡大し、MCS製品の自然な機能セットと一致させるように努力することの利点は、当初BIOSチームや経営陣にとって明確ではなかったが、時間の経過とともにそうなっていった。BIOSチーム以外の依存関係に対処することで

生じる障害の透明性の向上、効果的なレトロスペクティブの結果、各新スプリントで有意義な実験ができたこと、そして私自身が継続的に指導したことなど、さまざまな要因によって意識の高まりがもたらされたのです。詳細は、後述の_BIOS Component Boundary Expands_のセクションに記載されています。

奇妙なことに、LeSS チームの拡大したマルチコンポーネント境界は、製品アーキテクチャ全体の難解なコンポーネントの奥深くから始まり、その後、ユーザーインターフェースに向かってゆっくりと拡大していきました。これは、LeSS の採用が始まる場所としては珍しく、理想的とは言えませんが、経営陣の賛同と変化への関心が見出された場所でした。たとえ理想的とは言えないスタート地点であっても、チームの創造的な可能性を引き出すだけで、どれだけのことが成し遂げられるか、驚くべきことなのです。

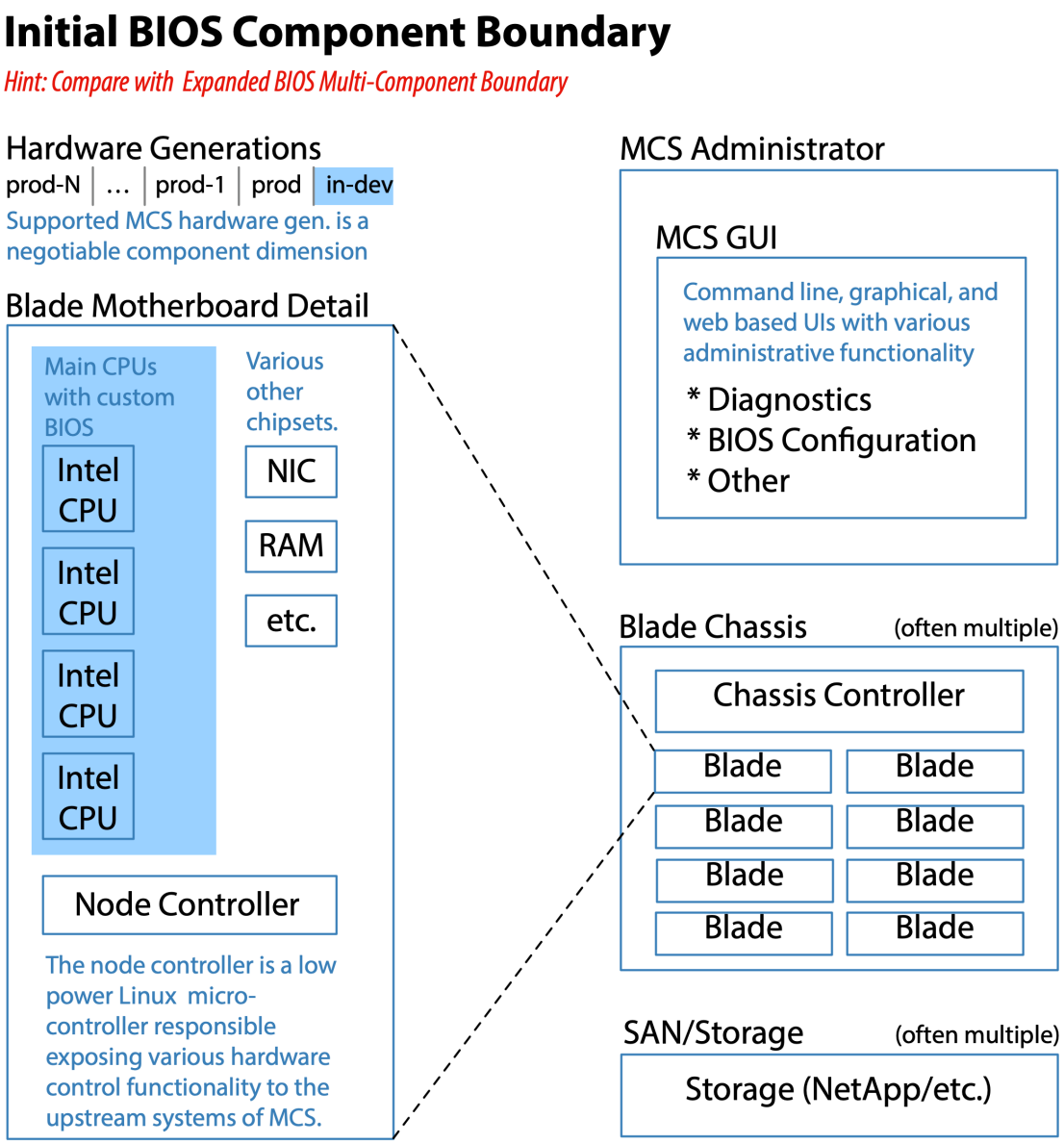

図 8: 初期の BIOS コンポーネント境界は、カスタム BIOS のみを含んでいました。この狭い範囲でも、カスタム BIOS 自体の中に何百もの特殊なコード領域があり、何百万行ものCコードが含まれていました。数十人の BIOS エンジニアのうち、BIOS コードの 1 つか 2 つ以上の側面を知っている者はほとんどいませんでした。

Expanded BIOS Multi-Component Boundary

Hint: Compare with Initial BIOS Component Boundary

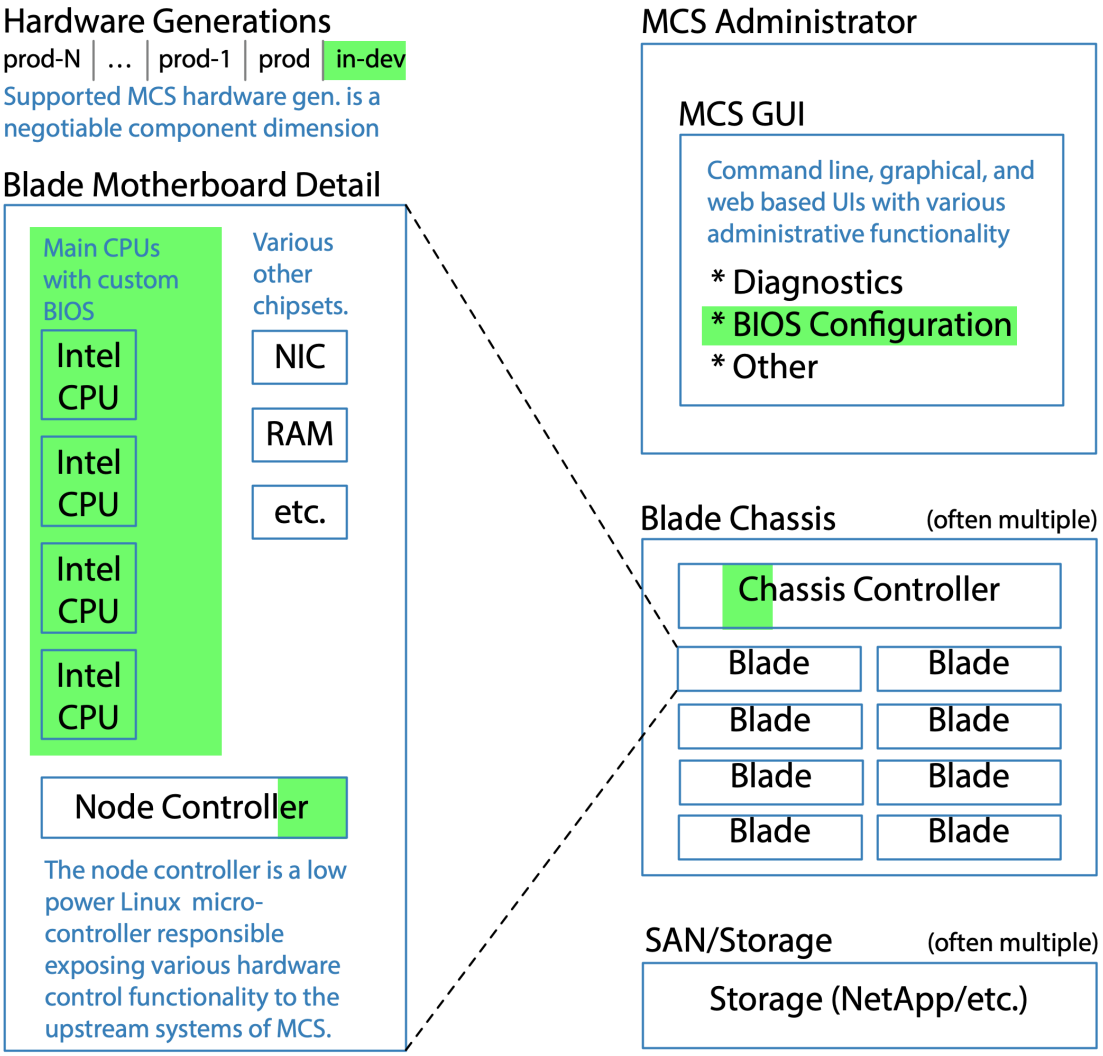

図 9: 拡張された BIOS マルチコンポーネント境界は、BIOS 構成制御パスに沿ったすべてを含んでいました。これは自然な製品要件領域にマッピングされ、製品管理グループのプロダクトオーナーが容易に理解することができた。完全には実現しなかったが、この方向へ進む努力はなされた。

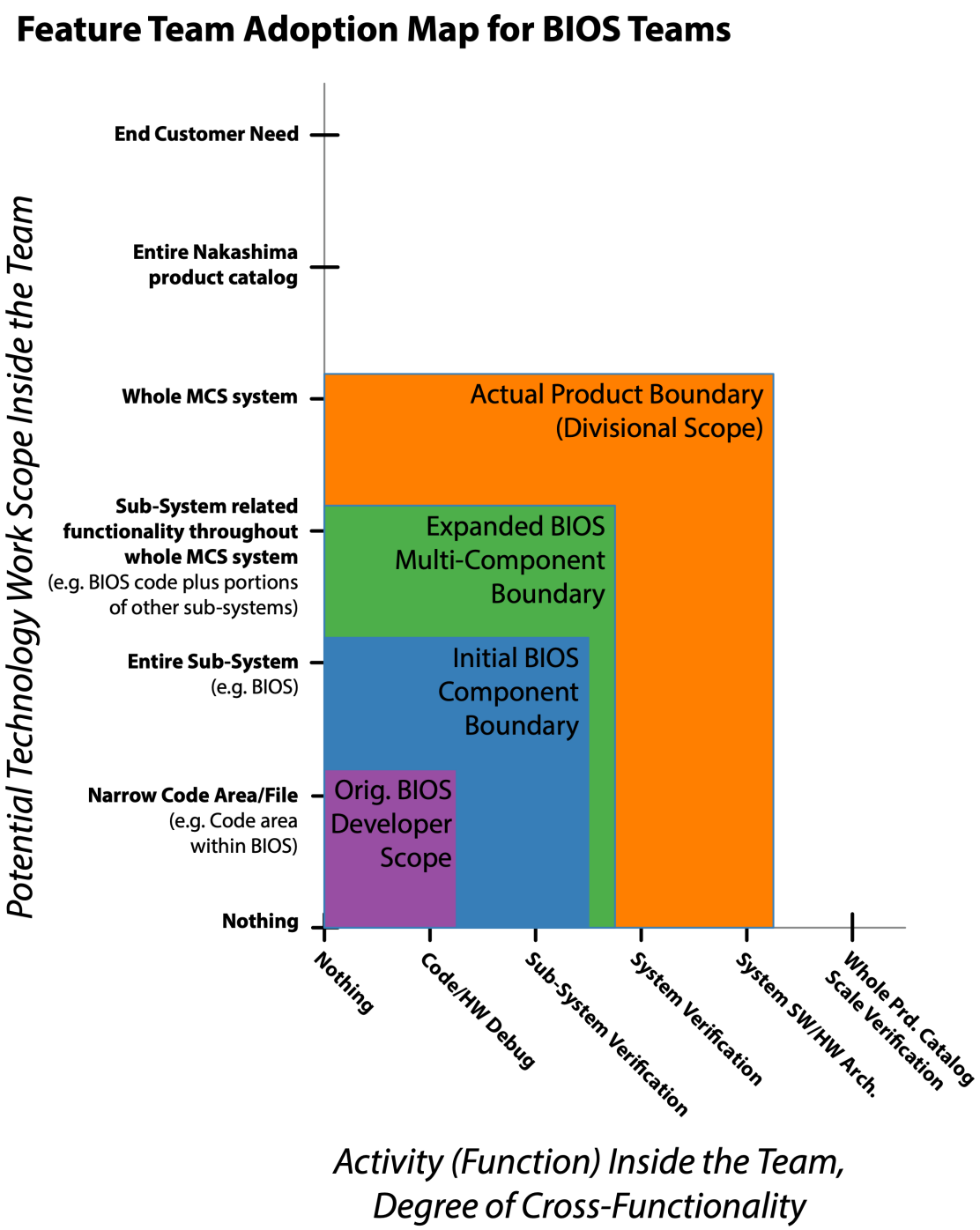

図 10：BIOS 開発者はもともと、BIOS のカスタマイズの狭い側面にそれぞれ特化した数十人の緩やかな集団という感じだったんです。BIOS システムだけでも数百万行のコードがあり、非常に難解なシステムドメインでした。

BIOS チームの中間で

新しく結成された BIOS チームにふさわしい形容詞を選ぶのは、ちょっと難しいですね。

新しく結成された BIOS チームは、レガシーBIOS チームよりもはるかに多くの**クロスファンクショナル**でした。各チームは、BIOS の変更に対して責任と権限を持ち、エンドツーエンドのテストを行うことができました。

新しく結成された BIOS チームは、最初から**セルフマネージメント**と**共同ロケーション**を採用していました。しかし、レイオフやその他の組織変更により、ミーチャと私が意図したほどには**長続きはしませんでしたが。

大規模スクラムの_フィーチャーチーム Adoption Maps_ガイドには、チームを説明するための有用な定義がいくつかあります。そのうちの 2 つを紹介します。

> **拡張コンポーネントチーム** — コンポーネントに関する作業スコープは限定的であるものの、より広範な製品全体の中で自らの担当部分が正常に機能することを確認する責任を担うチームは、すべて「拡張コンポーネントチーム」にあたる。
>
> **フィーチャーチーム**とは、製品全体に焦点を当て、顧客中心の機能の明確化からそのテストに至るまで、一連のプロセスに関与するあらゆるチームを指します。フィーチャーチームのあり方には幅があります。単に、必要とされていると明示された機能の実装のみに限定して活動することもあれば、製品の定義が十分に広範である場合には、顧客が抱える真の課題を特定・解決することに関与し、ひいてはシステム全体にわたる製品の共創を担うこともあります。

新しく構成された BIOS チームは、結成当初から「拡張コンポーネントチーム」として活動することが可能でした。彼らは、自分たちが行ったあらゆる変更について、エンドツーエンドのテストを行う能力と責任を持っていました。そのテストは、MCS GUI（そのコードは書いていない）から、シャーシコントローラ（そのコードは書いていない）、ノードコントローラ（そのコードは書いていない）を経て、最終的に Intel CPU 上で動作する BIOS コードに至るまで拡張された。一方、新しく構成された BIOS チームのコーディングは、当初は図 8 で説明した「Initial BIOS Component Boundary」内だけであった。これは、「拡張コンポーネントチーム」の定義と完全に一致することがおわかりいただけるだろう。

新しく構成された BIOS チームは、当初はフィーチャーチームの定義に合致しなかった。フィーチャー・チームになるには、関連するすべてのコードをフロントからバックまで含める必要があります。私がナカシマ MCS に在籍していた頃は、BIOS チームがこれを達成することはなかったが、少しずつ前進していた。

新体制の BIOS チームをまとめると

1. 拡張コンポーネントチームとしてスタートし、セルフマネジメントと共同ロケーションを実現。
2. BIOS コンポーネントと MCS GUI コンポーネントの間の通信経路に沿ったより多くのコンポーネントをカバーするために、コンポーネント拡張チームとなることを目指した。
3. エンドツーエンドの機能を提供するために必要な「すべて」に責任を拡大し、「フィーチャーチーム」になることを計画した。

BIOS の組織的な背景

BIOS の LeSS 指向の採用は、役員層の入れ替わりが激しい時期に行われました。BIOS の採用が始まったとき、すべてのハードウェア開発を担当する副社長が非常に協力的で、組織全体の他の様々な副社長も同様だった。これは、最初の採用活動を可能にし、短期的に成功させるのに十分であった。しかし、SVP や副社長クラスの経営陣が交代すると、私たちの努力は関係者の多くから成功したと認められていたにもかかわらず、必要なサポートの多くを失ってしまったのです。新しい SVP が両義的な態度をとり、ハードウェア担当の VP を失い、さらに大規模なレイオフが行われたため、最終的に我々の努力は水の泡となりました。次の一連の組織図が、このような状況を浮き彫りにしています。

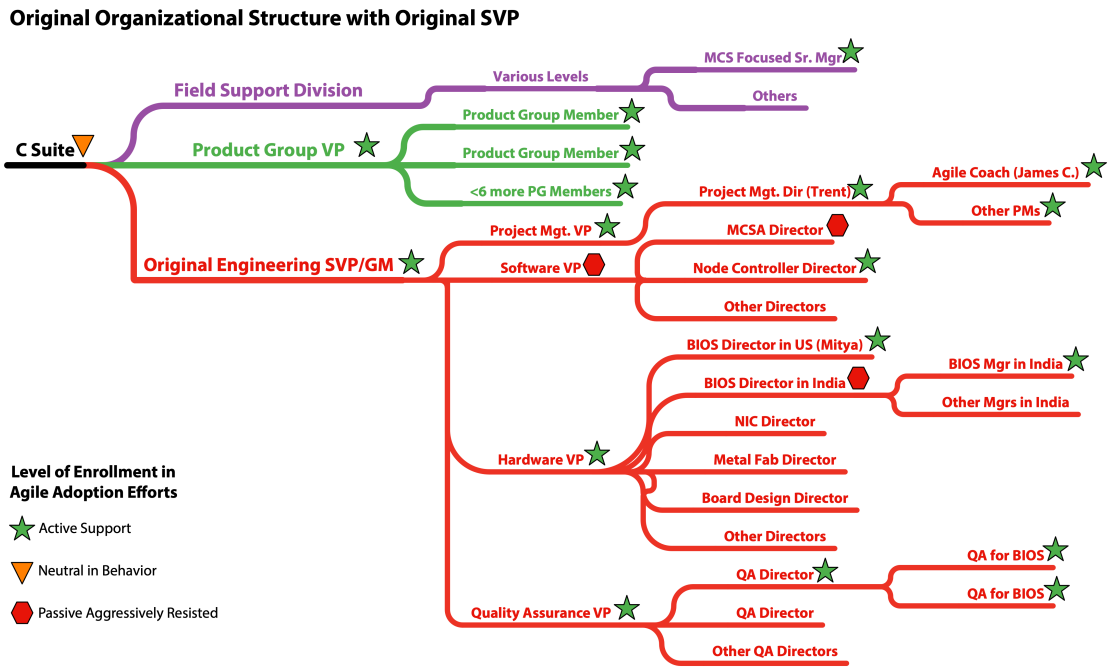

図 11：MCS 部門のエンジニアリングの最初のシニア VP/GM は、アジャイル採用の取り組みに非常に協力的でした。また、組織の多くで積極的なサポートを受けることができました。私が組織をよりよく理解し支援しようとしたとき、多くの役員、マネージャー、および個々の貢献者が積極的に指導してくれた。残念ながら、この上級副社長の在任期間は非常に短く、製品のより純粋なソフトウェア部分を担当する主要な副社長が、真の変化に対して消極的かつ積極的に反対していました。

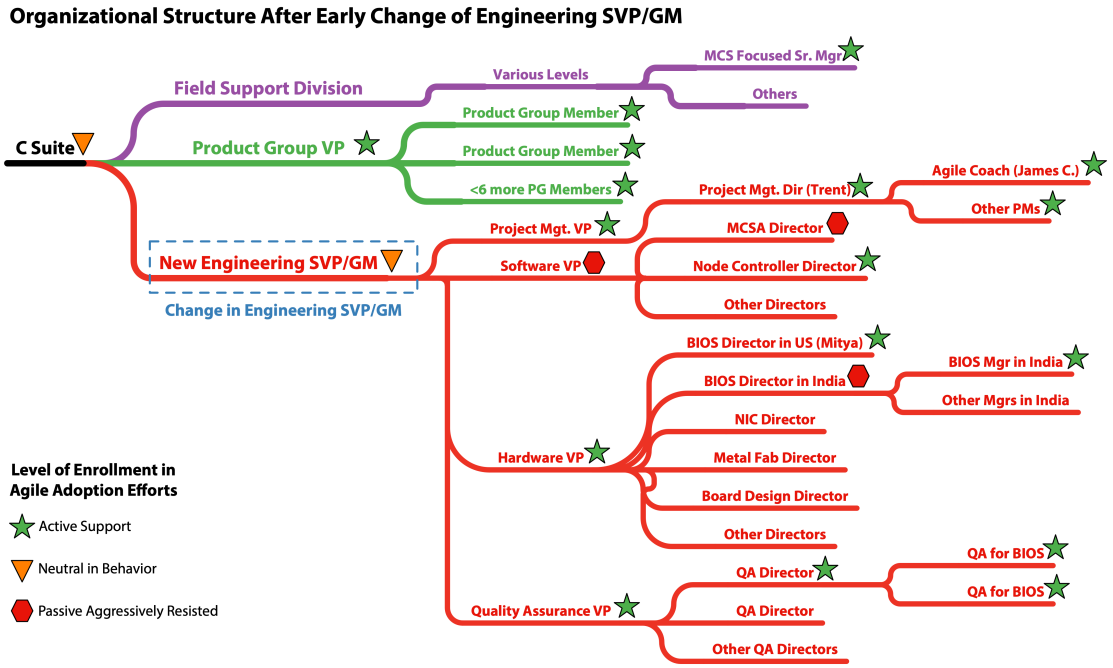

図 12：MCS 部門のエンジニアリングの最初の Sr. VP/GM は、私が着任する前の数ヶ月間だけその役割を担っていました。私が着任して数カ月もしないうちに、彼は別の上級副社長に取って代わられました。今にして思えば、この新しい副社長の C スイートからの指示は、事業部の規模を合理化することであったのは明らかである。この新しい上級副社長は、私にはほとんど何も言わず、アジャイル変革の取り組みに積極的に関わろうとはしませんでした。概ね私の目の届かないところで起きていたことではあるが、私はプロジェクトマネジメント担当副社長から積極的な支援と風通しの良さを受け続けていたと思う。新しいエンジニアリング SVP が引き継いでから、時間の経過とともにいくつかの追加の組織変更があったが、ハードウェア VP が去るまで、アジャイル導入を試みるチームにとってあまり重要なものはなかった。

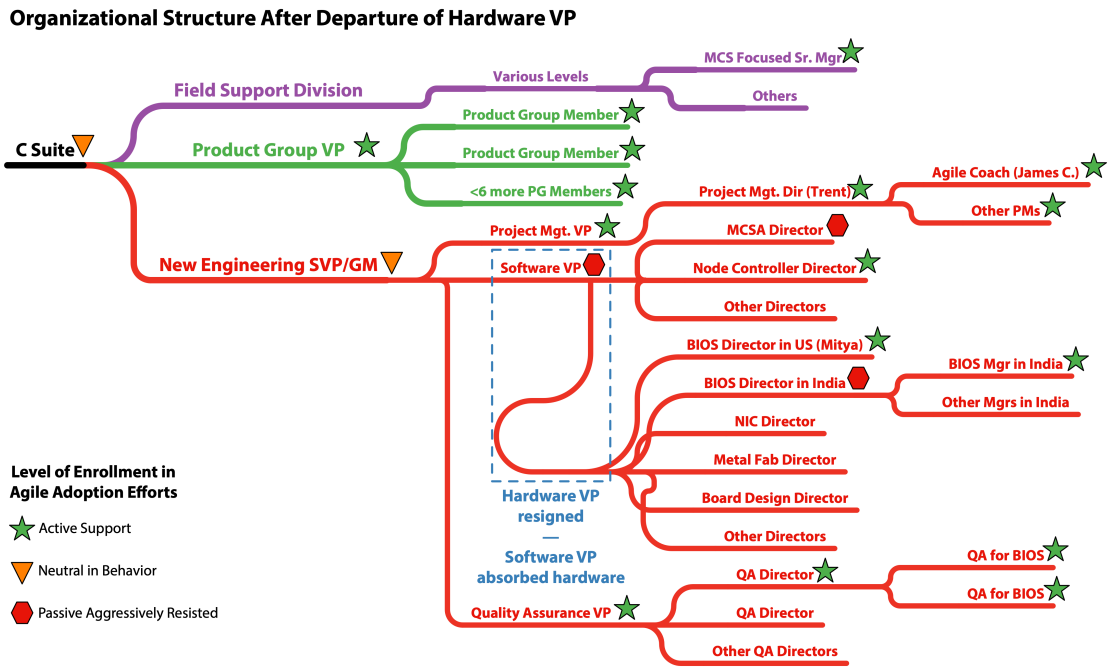

図 13：近々予定されていた積極的なレイオフの雲の下で、多くの人々が自発的に組織を去り始めた。同じ頃、資金力のある新興企業が、MCS 部門の優秀なエンジニアや管理職を積極的に採用し始めていました。その中で、BIOS チームが所属していたハードウェア担当の副社長も離職しました。新しいエンジニアリング SVP は、ハードウェア VP を補充するのではなく、これまでハードウェア VP を通じて報告していた人たちを、ソフトウェア VP を通じて報告させることを選択しました。ソフトウェア VP は常に受動的で積極的にアジャイル導入の努力に反対していたので、これは良い兆候ではありませんでした。数ヶ月の間に BIOS チームメンバーの半分が解雇され、私の契約は終了し、ミーチャはハードウェア VP を追って、ハードウェア VP が去ったのと同じ資金力のあるスタートアップに移りました。それから 1 年余り、トレントもナカシマコーポレーションを去りました。

BIOS コンポーネントの境界（バウンダリー）と地層

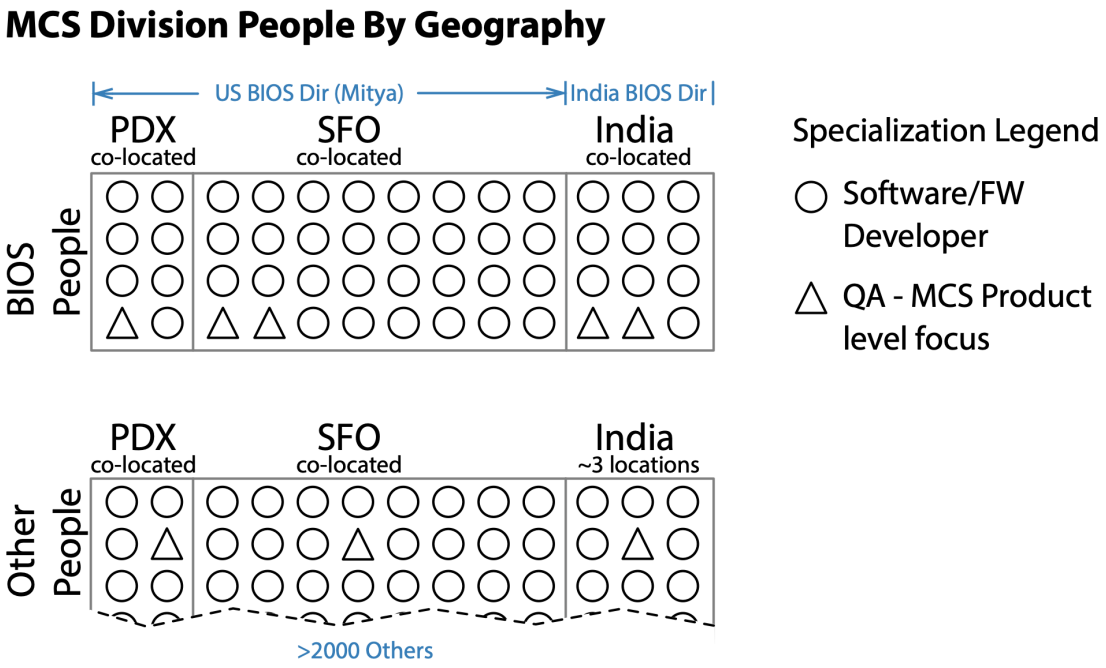

図 14： MCS 事業部内の大部分の人は、各都市の 1 つのビル内の 1～2 フロアに同居しています。私たちは、BIOS チームメンバーが一般的にチームメイトの数フィート以内に座っていることを確認するために注意しました。ワークステーションは、一般的にスウォーミングアップに適した環境でした。作業の半分はラボスペースで行われたため、多くのチームメンバーは事実上 2 つの作業場所を持つことになりました。アメリカでは、一部のテスト担当者を除き、全員がミーチャを通じて報告を行っていました。テスティングスペシャリストも、同じ場所にいて、BIOS に完全に割り当てられ、チームの他のメンバーと同じように扱われた。BIOS の開発者の中には、1 人か 2 人分散している人がいましたが、それは唯一の例外でした。

残念ながら、私たちはマトリクス構造の一部として、スペシャリスト・マネージャーの役割を公式に取り除くことはできませんでした。それでも、マネージャーは専門性を強調したり、それにしがみついたりせず、人々が多能工になることをサポートしました。

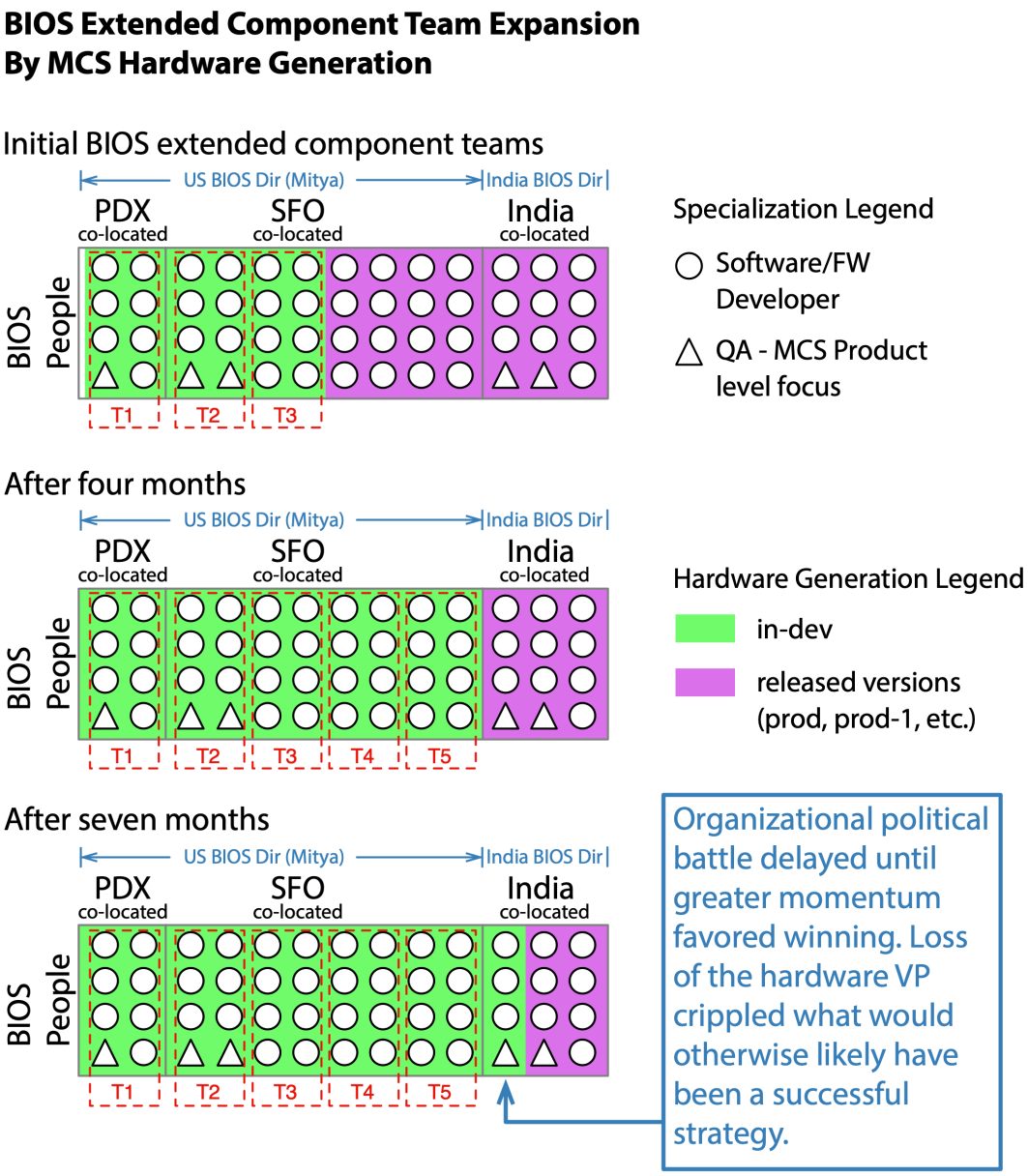

図 15: _MCS ハードウェアの世代は、BIOS コンポーネント内の LeSS 指向の採用の境界を定義する際に、コンポーネント次元の一つとして使用されました。ハードウェア担当副社長のサポートとインドへの数回の出張があれば、ミーチャと私は政治的な問題をうまく解決できたと思われます。残念ながら、副社長層の変化とレイオフがこの戦略を阻みました。タイムラインと組織構造図にある情報は、この図に書かれていることに関連しています。

歴史的な理由により、ハードウェアの世代が変わるたびに、BIOS のコードベースが完全に分離され、共通機能を再利用可能なサブコンポーネントにほとんど集約されないという現実がありました。

興味深いことに、これはBIOSチームをより持続可能なペースで成長させる機会を提供するものであった。以前は関与していなかったBIOS開発者が、以前の生産ハードウェア世代に関連する不具合の解消に取り組み始めると、LeSS指向の構造内で、さらに機能横断的なチームを結成することになりました。BIOSコードベースのダイナミクスとAMIとの関係については、さらに詳しく説明します。

このようにして、ミーチャを通して報告する米国に拠点を置く人々と機能横断的なBIOSチームを立ち上げるのに半年間の猶予を得た後、インドのBIOS開発者と管理者をより積極的に巻き込む必要が出てきました。インドのBIOSエンジニアが米国のBIOSエンジニアと同じBIOSコードベースで作業を始める頃には、米国のBIOSエンジニアは、まだ機能横断的なBIOSチームに入っていないインドのBIOSエンジニアが同じ改善された品質基準を満たさない場合に、反発できるほど結束していたのです。

最初の苦労は、インドに拠点を置くBIOSエンジニアが、クリーンなビルドと基本的なスモークテストをパスしない限り、コードをチェックインしないように注意する、というような単純なものであることが多かった。追加の自動テストと、機能横断的なBIOSチームの直接管理下にあるプライベートCIサーバーは、改善された品質基準に従わない開発者によって問題が発生した場合に、米国ベースのチームがそれを検出するのに役立ちました。

機能横断的なBIOSチームの成功により、ミーチャと私は、根本的な構造的問題の解決に着手するための政治的立場をより強固なものにすることができたのです。ミーチャは、プロジェクトマネジメントSVPの要請で、機能横断的なBIOSチームの改善についてナカシマ全体にプレゼンテーションを行っていました。

インドにいるBIOSチームへの解決策は、機能横断的なBIOSチームを1つまたは2つ追加することだった。理想を言えば、こうした構造的な問題はすべて最初から解決しておきたかったのですが、当時はまだ政治的に実現可能ではありませんでした。その代わりに、私たちは実現可能なことをしたのです。私たちは、将来的に政治的な勝利が得られるよう、意図的に問題を分離し、遅らせたのです。私たちは、2つの別々の運営モデルから生じる摩擦を予期していただけでなく、将来的に意味のある構造改革を政治的に実現するために、それを当てにしていたのです。

ミーチャと私は、インドに行く準備をし、政治的な解決に乗り出そうとしていました。もしハードウェア担当副社長が残っていたら、BIOSコンポーネントという境界の中で、より差し迫った政治的課題を解決することができたと思うのです。すでにインドにいるBIOSマネージャーを味方につけていましたが、ディレクター層でいくつかの問題を解決する必要がありました。

コードベースを分けることによる弊害

新しい世代のハードウェアが出るたびに、まったく別のBIOSコードベースを持つというのは、いいことではありません。私たちはその事実をうまく利用しましたが、望ましいこととは程遠かったのです。

コピーペーストの再利用があまりにも多く、開発者が自信を持って後方互換性のある変更を行えるようにするための自動テストカバレッジが不十分で、AMI コードベースの変更から MCS 関連の変更を分離するために AMI プラグインレイヤーを使用することに失敗していたのです。

このような自業自得の負担により、インテル® CPU の変更に対応することは、他の方法よりもはるかに困難なものとなりました。皮肉なことに、インテルの CPU の変更に対応する能力は、インテルのリリースサイクルよりも速く、MCS 部門の収益と市場シェアを維持するために非常に重要である。

これらの問題を解決するための努力は、後述する「BIOS の定義」の項で見ることができる。

BIOS 地理的に分散したチーム

グローバルに最高レベルの価値を提供するために、最高レベルの適応性を最適化するという観点からは、これほど多くのエンジニアが製品に携わることはなかったはずだ。また、エンジニアが複数の場所やタイムゾーンに散らばっていることについても同じことが言える。しかし、最初からそうであったのだから、そうでないことを納得させようとしても、うまくいくはずはない。

Intel の発売日に間に合わせるという重要性と、レガシーコードベースにある膨大な量のコピーペーストの再利用が相まって、困難な状況を作り出していたのです。米国に拠点を置く BIOS チームのメンバーのほとんどが力を合わせなければ、必要な変更を時間内に行うことができず、また、この状況を引き起こした不十分なエンジニアリング手法に対処しながら行うには、十分な知識を結集させることができなかったでしょう。それに対して、まったく異なるタイムゾーンにある別の地域を追加することは、事態を遅らせるだけである。

LeSS の本からいくつかの異なる引用が、この点を強調するのに役立ちます。

> 「まずは少数の優秀なメンバーでチームを立ち上げ、増員は『本当に人手が足りず、痛みを伴うほど切迫した状況』になってから行うべきだ」――こうした理想的な展開が、実際に起こることは極めて稀である。『Scaling Lean & Agile Development』より

> 「そもそも、これほど多くの人員が必要なのか？」といった議論に終始するのではなく、私たちはアジャイルやリーン開発の原則を活用して開発プロセスを改善できるよう、現場の人々を支援することに注力している。そうすることで、やがてチーム自身が「あちこちに人員を過剰に配置しすぎている」という事実に、自ずと気づけるようになるからだ。『Practices for Scaling Lean & Agile Development』より

品質保証グループという名前が最悪だ

品質とは、後付けできるものではないことを認識することが重要です。なぜか？製品の品質は、製品開発システム全体の成果であり、品質管理部門が個別に責任を負うものではありません。テストグループの正式名称は「品質保証」だが、より正確には「品質管理」というべきだろう。

品質保証の正しい使い方は、テストファーストまたはテスト駆動開発の確立、効果的なコードレビューの実践、静的コード解析ツールの設置、継続的インテグレーション行動の確立、ペアリングまたはスワーミング行動の確立などの活動を指すものです。これらのプラクティスは、ソフトウェア開発が「コードコンプリート」とみなされた後に品質をボルトで固定しようとか、その他のナンセンスなこととは一切関係がないことに注意してください。

品質保証グループは、実にお粗末な名前であった。その名前からして、高品質な製品を開発するためのメンタルモデルが崩れていることがわかります。しかし、歴史的な正確さを期すために、私はレガシー構造の中ですべてのテスターを含む組織ブランチを QA グループと呼び続けることにします。このドキュメントを読むときは、この区別を覚えておいてください。

BIOS テスターがもたらしたエンドツーエンドの知識

MCS の幅広さと複雑さ、そして過剰な専門性の歴史は、それまでレガシー構造の中でエンドツーエンドのテ ストにほとんどの時間を費やしていたテスターほど、エンドツーエンド製品を理解している人がいないことを意味し ていました。その結果、テスターは、従来の BIOS ファームウェア開発者と同じように、それぞれの BIOS チームに価値をもたらすことになったのです。

テスト出身の BIOS チームメンバーは、品質保証担当副社長を通じて、正式な報告を行っていた。彼女のサポートと関連する QA ディレクターのサポートは、BIOS チーム内での事実上の平等を可能にするために重要であった。この正式な報告関係の維持は、BIOS の LeSS 指向のチームが生み出す品質の抜本的な改善を組織として受け入れてもらうために有効であった。

テスターのために別の報告構造を維持することは、組織的な脆弱性をもたらしました。品質保証担当副社長の役割が変われば、起きているポジティブな変化の多くを簡単に解きほぐしてしまう可能性がある。長期的な目標として、組織構造における正式なテスト特有の区別をなくすことは、テストの経歴を持つチームメンバーが、そのチームのスプリントバックログとは無関係の仕事に気を取られるリスクを回避するのに役立つだろう。

リーン&アジャイル開発の規模拡大のための実践」の第 3 章では、実践と戦略の両レベルでテストに最も適したアプローチを行う方法について包括的に扱っています。

拡張 BIOS マルチコンポーネントの目標と制約

電子メール、ホワイトボードの走り書き、口頭での会話以外に正式に文書化されたことはありませんが、BIOS LeSS 採用の目標には以下が含まれます。

- MCS ハードウェアの将来の変更に MCS BIOS を適応させる能力を向上させ、特に Intel チップセットへの定期的な更新を行う。

- 技術的な卓越性を評価し、「契約ゲーム」によって生じる非生産的なストレスを回避する文化を創造する。

最新のインテル製チップセットの BIOS サポートは、MCS 製品と競合他社の製品との差別化の重要なポイントになることはほとんどありませんが、インテルのリリースイベント時に同等でなければ、一晩で市場シェアと収益を大幅に減少させることになります。このことは、経営陣もよく理解していた。そのため、次のような制約が重要であった。

- BIOS のアジャイルな採用努力が、MCS 製品が Intel CPU の製品版をサポートできない理由となることは、決して許されない。

その結果、BIOS の適応性の目標は、ミーチャのように上級管理職の頭の中にあるわけではなかったが、その目標の重要性は暗黙のうちによく理解され評価されていた。

このような共通認識にもかかわらず、従来のレガシー・ウォーターフォール・アプローチの機能不全の程度は、広く受け入れられてはいなかった。コントラクトゲーム」、「お粗末なエンジニアリング手法」、「過剰な専門化」などは、部門全体で観察することができた。

上記の目標について考えてみると、「大規模スクラム」で述べた「組織的完璧性ビジョン」のガイドにある LeSS 完璧性ビジョンと非常に強く共鳴していることに気づくでしょう。また、そのガイドにある、ローカルな最適化と本当のシステム改善を見分けるための 2 つの質問とも強く共鳴していることがわかるでしょう。

上記の目標をサポートする BIOS グループ内の目標には、以下のようなものがありました。

- 最も価値の高い BIOS の作業をより見えるようにするため、単一の BIOS コンポーネントバックログを確立する。単一の製品レベルのバックログに向けたベイビーステップ。
- 各エンジニアの知識とスキルを拡大し、製品のより広い部分を網羅することで、各チームの適応性を高め、新たに発見された最も価値の高い作業に切り替えて集中する全体的な能力を向上させることができます。
- 各 BIOS チームによる「製品全体への注力」（LeSS の原則）を強化し、MCS の大部分にまたがる拡張 BIOS マルチコンポーネント境界に向かって移行します。
- MCS BIOS コンポーネント周辺の職人技の実践を改善するよう努力する。

- コントラクトゲームを可能な限り回避する。

ダイアグラムだけでわかる BIOS 採用ストーリー

タイムライン図、コンポーネント境界図、機能採用マップ、組織図、ハードウェア世代別機能チーム拡大図とそれぞれの図のキャプションを合わせて、ストーリーアーチの全体像を理解するのに十分な文脈を得ることができます。

BIOS エンジニアリングと文化的課題

MCS の中でカスタマイズ BIOS システムが果たす役割について、大まかな理解が得られたと思います。ここでは、BIOS チームが直面する技術的・文化的な課題について、深入りしないようにまとめたいと思います。もし、この内容が気になるようであれば、読み飛ばしていただいて結構です。

AMI は BIOS コードの基礎を提供した

MCS 計算ノードのカスタム BIOS は、AMI 社からライセンスされた常に更新されるコードベースを修正することで作成されました。AMI が BIOS コードベースの継続的な変更を行う典型的なイベントには、以下のものがあります。

- Intel が、プロトタイプのチップセットやリファレンスボードの設計において、ハードウェアとファームウェアのインターフェイスの一部を変更した場合。
- AMI は、プロトタイプのインテル製チップセットまたはリファレンスボード設計の改良された機能によって実現される、新しい BIOS 機能のサポートを追加します。
- AMI またはライセンスパートナーの 1 社が特定したバグを修正する。
- AMI が理にかなった理由で BIOS コードベースを変更することを決定した場合。

MCS BIOS カスタマイズと AMI コードベース間のインターフェイスがほぼ安定しているため、これらの通常の AMI アップデートの影響は最小化されるでしょう。しかし、実際にはそうではありませんでした。AMI コードベースとの正式なプラグインは存在するものの、ナカシマの BIOS カスタマイズのほとんどは、歴史的に AMI コードベースの奥深くで行われてきたのである。

MCS の BIOS エンジニアは、新しいチップセットを作るたびに、古いチップセットの BIOS バージョンに合わせたカスタマイズから、新しいプロトタイプチップセット用のコードに機能をコピーすることを試みた。同様に、MCS の BIOS エンジニアは、AMI のコード変更を常に把握し、それらをアクティブなワーキングブランチにマージし、手作業で再テストを行う。テストには、ラボでの実作業が必要なものもあったが、これまでよりもはるかに多くのテストを自動化することが可能であった。

デスマーチ文化はインテル・リリースに支えられている

市場の流れは、インテルのチップセット発売日に支配されています。ハードウェアインテグレーターは、プロトタイプチップセットとリファレンスボードアーキテクチャにいち早くアクセスすることができます。インテルのリリース日に合わせて出荷する能力は、市場で生き残るために不可欠です。

インテルの仕様変更に対応できるように最適化することが、組織設計上の賢明な判断であった。しかし、残念なことに、従来の組織の対応は、問題に大量の人員を投入し、リリース日に向けてデスマーチすることでした。そのため、発売日が迫ってくると、どうしても品質が犠牲になってしまうのである。

ナカシマ MCS の知の喪失

ナカシマの MCS を構成するパーツを最初に作った人たちの多くは、何年もかけてナカシマを去っていった。そのような人たちは、部族の知識も一緒に出て行ってしまったのです。理想は、テストピラミッドの全レベルで広範な自動テストが行われ、よく練られた読みやすいコードが存在することです。最低でも、システム全体のアーキテクチャを詳細に説明する有用な文書があるはずです。ウォーターフォール文化の納期のプレッシャーの中で開発された複雑な製品に期待されるように、これらのものはほとんど存在しませんでした。

Intel BIOS は高度に専門化されている

AMI は、BIOS のカスタマイズに精通したエンジニアは、世界中に数千人しかいないと見積もっています。x86 ハードウェアファームウェアインターフェースの動作やデファクト仕様の多くは、PC 革命の初期に遡る部族的知識を必要とします。

実際には、MCS BIOS の各カスタマイズ領域は、1 人または 2 人のエンジニアが AMI コードベースを深く掘り下げ、そこで見つけたものをリバースエンジニアリングした結果です。ある意味、これはプロのソフトウェアエンジニアが一日かけて行うことと変わりませんが、違いは、BIOS 開発において、これがいかに難解で頻繁であるかということです。

3 つの地域にまたがる多数の技術者

40 人ほどの技術者が忙しく働けるほどの仕事量である。職人技を磨けば、最終的にはもっと少ない人数で済むはずだ。とはいえ、自業自得の技術的な問題から脱却するには、膨大な作業と調整が必要だった。

歴史的な理由により、BIOS にフォーカスしたソフトウェア、ハードウェア、テストエンジニアが、ポートランド、サンフランシスコ、ベンガルールといった大都市に散らばっていました。チームの規模を拡大する計画では、この分散問題を解決しなければなりません。幸いなことに、私たちは同じ場所にいるチームに自己組織化することができま

した。さらに、ベンガルールで行われている作業は、ポートランドやサンフランシスコでの作業からある程度独立していました。

大きな技術的課題

MCS BIOS チームが直面した課題の多くは、自ら招いたものでしたが、いくつかは、仕事そのものに内在するものでした。それらは以下の通りです。

- プロトタイプのチップセットとボードに対するハードウェアのアップデートは定期的に行われます。新しいボードデザインの製作にかかるリードタイムは、通常6週間程度でした。これは、物理的な問題ではなく、組織的な障害であると言えるかもしれません。

- プロトタイプのチップセットは、ファームウェアと同じように問題の原因となる可能性が高かったのです。その結果、テストとトラブルシューティングの重要な側面は、研究室での実地作業を必要としました。

- x86 BIOS ツールチェーンは、非常に古く、使いにくいものです。これは、AMI がツールを改善するために大きな投資をする経済的動機が不十分であったためです。

- 自動ユニットテストのようなソフトウェアエンジニアリングの実践は、世界全体の x86 BIOS 開発文化の中で一般的に欠けています。この文化的な歴史は、MCS BIOS チームがカスタマイズした AMI コードベースでもはっきりと観察することができた。このため、以下のような複雑な問題が発生しました。

    - 単体テストツールは、すべて手作業で行う必要がありました。
    - AMI BIOS コードベースに効果的な依存関係逆転メカニズムを組み込むことは、通常の Java や C#フレームワークで行うよりもはるかに困難だった。
    - 書籍、記事、フォーラムの投稿、サンプルコード、その他のリソースで、BIOS コンテキストでのユニットテストのガイダンスを提供するものがなかった。
    - 自動ユニットテストのコンセプトはもちろん、テストファーストや TDD の実践も、MCS BIOS エンジニアにとって完全に異質なものでした。

- 低レベルファームウェア開発の世界では、すべてが平均的なソフトウェア開発の努力よりも退屈である。MCS BIOS に関連するいくつかの例を挙げよう。

    - オペレーティングシステムが存在しないため、オペレーティングシステムサービスが存在しない。
    - コードがクロスコンパイルされ、その後ターゲットにフラッシュされなければならない。

- BIOS ブートプロセスの初期段階では、TCP/IP ネットワークスタックが存在しないため、ターゲットとの通信が特に困難である。ターゲットデバイスとの初期の通信は、シリアルポート通信と同様の制約のあるメカニズムに限られていました。
- ファームウェア開発エンジニアの多くは、ソフトウェア工学の専門知識よりもマイクロエレクトロニクス工学の専門知識を持つ人が多い傾向にあります。

ここに挙げた問題は、いずれも克服できないものではありません。多くの場合、大規模な Web アプリケーション開発など、他の領域の技術が、これらの問題の多く を解決するためのヒントを既に与えてくれています。

例えば、Java と C#はどちらも洗練された依存性注入フレームワークを持っており、既製品として利用することができます。James Grenning による_Test Driven Development for Embedded C_は、これらの技術の相互提供を改善する試みの一例である。

BIOS 採用の取り組み

ミーチャと私は、自分たちの取り組みが大変なものであることに気づきました。BIOS 機能チームの立ち上げ、立ち上げ、成熟化の手順は、診断チームで行ったこととほぼ同じでした。主な違いは、課題の難易度と規模です。

大きな違いは、どちらかというと単純なスケーリングの側面に関わるものでした。

BIOS 起動手順

ミーチャと私は、まずサンフランシスコとポートランドのエンジニアに焦点を当てることにしました。サンフランシスコとポートランドは、新しい Intel チップセットとボード設計の立ち上げに主に集中していました。ベンガルールのエンジニアは、すでに生産されているチップセットのサポートに関わる問題の修正が主な仕事でした。

私はミーチャと一緒にトレーニング会場を手配し、サンフランシスコとポートランドの MCS BIOS エンジニア全員にスクラムトレーニングを実施するスケジュールを組みました。その際、他のグループと密接に協力しているテストエンジニアや、その他の関係者も数人参加させるようにしました。私の記憶が正しければ、この結果、3 週間かけてサンフランシスコで 2 回、ポートランドで 1 回のトレーニングセッションが行われました。ミーチャは、サンフランシスコの各トレーニングセッションに参加しました。ポートランドのトレーニングには、ミーチャは参加しませんでしたが、クリシュナ・ミシュラがサポートに入りました。ミティアはポートランドのトレーニングには参加しませんでしたが、クリシュナ・ミシュラがサポートに入りました。ミティアは、その 1 週間後に行われたローンチの際、私と一緒にポートランドに向かいました。

立ち上げ作業は、サンフランシスコとポートランドで分担して行われた。最初の 2 日間は、木曜日と金曜日にサンフランシスコで行われ、ポートランドの BIOS チームのメンバーが会議に参加しました。翌週、ミーチャと私はポートランドで合流し、ポートランドのチームメンバーと共に立ち上げ作業を続けました。これにより、ポートランドのチームメンバーは、前週の作業に影響を与え、調整し、批准する機会を得ることができました。ポートランドの立ち上げ活動では、サンフランシスコのチームメンバーとも頻繁に打ち合わせをしました。

その際、私は達成すべきことをリストアップし、全員が各項目の意図を理解していることを確認しました。そして、必要なときにだけ参加することで、できる限りグループの主導権を握れるようにしました。このリストには、BIOS コンポーネントのバックログの作成、共通の Done の定義、個々のチームの境界線の決定、イベントのスケジューリングなどが含まれています。

BIOS コンポーネントバックログ

この立ち上げ作業で最も興味深かったのは、コンポーネントバックログの作成です。次の Intel チップセットを立ち上げるために必要な知識は、チーム全体に非常に広く分散していた。どのチームメンバーも、全体をよく理解していなかったのだ。この解決策には、数日間のブレーンストーミングと、小規模なワーキンググループでの個々の PBI の詳細な改良、そして最後に段階的なストーリーマッピングの取り組みが必要でした。

初期コンポーネントバックログのブレーンストーミング

最初の BIOS コンポーネントバックログのブレーンストーミングの構造はかなり標準的なもので、違いは達成された有用な詳細と調整のレベル、そしてそれをうまく行うために必要な時間の長さであった。

この場合、BIOS エンジニアは、過去に何度も集団で行ってきたことを行っていたのです。互いに必要な知識を引き出す作業を集団で行う限り、必要なものをほぼ網羅した膨大かつ詳細なリストを作成することが可能だったのです。今にして思えば、コピーペーストの再利用や手動テストに頼るなど、ソフトウェアのやり方が悪かったために、BIOS エンジニアは Intel のチップセットが新しくなるたびに、ほとんど同じ手順を再確認していたのだと思います。

私たちは、思いつくことをすべてポストイットに書いて、予約しておいた大きな会議室の窓ガラスに貼って、共同作業をしていました。そして、ミーチャや他のメンバーが、それぞれの付箋に目を通すように指示します。そして、その付箋紙を感覚的に思いつくままにグループ分けし、同時に重複するものを整理していきました。グループ内で各ポストイットについて議論し、コンセンサスを得ると、参加者はその場で新しいポストイットを作成し、それが意味をなすようにしました。その後、小グループに分かれて新しいポストイットを作成し、再びディスカッションセッションに戻り、すべてを統合しました。これが初日の大半を占めた。1 日目の終わりには、立ち上げ時のチェックリスト

の他の項目に切り替え、翌日のためにさらに多くのポストイットを用意させることに合意しました。

2 日目の朝は、コンポーネントバックのブレーンストーミングを続け、お昼頃にブレーンストーミングを終了しました。その後、ローンチチェックリストの他の項目の仕上げに戻りました。

最初の Sprint をサポートするのに十分な数のコンポーネントバックがありましたが、私たちが望み、可能であるとわかっている有用な詳細レベルを達成するにはまだ十分ではありませんでした。もし、これが典型的なソフトウェアエンジニアリングの文脈で開発されている新製品であれば、私たちが行ったことは無駄なことだと思います。しかし、この場合、私たちは長年にわたって蓄積してきた部族の知識を効果的に統合し、文書化したのです。

大規模スクラム」の「初期 PBR」ガイド。More with LeSS_にある_初期 PBR_ガイドでは、_初期プロダクトバックログリファインメント_をより詳細に扱っています。動機の 1 つは以下の通りです。

> 顧客中心的な見方に関する限られた知識。
>
> 古い項目が以前は顧客中心的な方法で表現されていたとしても、事前のサイロ化された専門家たちは狭いタスクに集中しているため、完全な顧客中心的な視点を理解していないのです。

リーン&アジャイル開発を拡大するための実践」の「第 5 章：計画」にも、関連するさまざまな実験があります。最も直接的に関連するのは、「Try…Kickstart large-scale Scrum with one initial Product Backlog refinement workshop_」である。

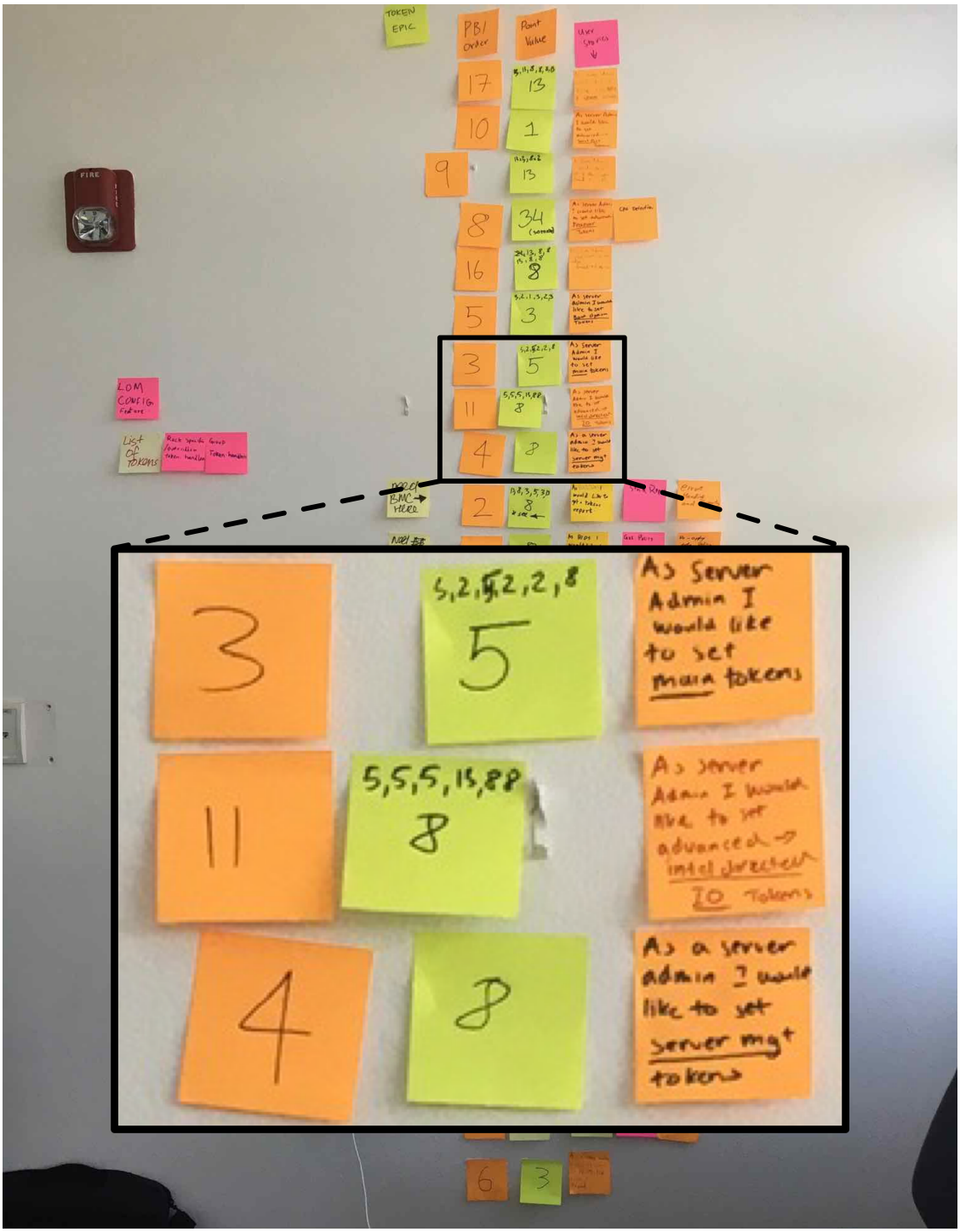

図 16：最初の BIOS コンポーネントバックログのブレーンストーミングでは、短い説明やタイトル以上のものはなく、一連の予備的な PBI ポストイットが作成されました。各 PBI ポストイットは、ポーカープランニングを使用して工数見積もりを行い、BIOS コンポーネントバックログの適切な順序を与えられました。出発前には、電子データへの移行に備え、作業風景の写真を撮りました。

初期 BIOS コンポーネントバックログのリファインメント

最初の 2、3 スプリントの間、私は、インセプションで作成された最初の BIOS コンポーネントバックログのさまざまな側面について最もよく知っているエンジニアの異なる小グループと一緒に働きました。私は、各ワーキンググループに INVEST テストを思い出させ、いくつかの PBI を作成する作業を手伝いました。一度、うまくまとまった PBI がどんなものかという感触を得ると、そのグループは、自分たちが知っているすべての PBI に手を入れ、洗練させていきました。ほとんどの場合、私は彼らと一緒にいて、彼らの作業をサポートしました。通常、各ワーキンググループは、1 行の説明をどのようにすれば良い受け入れ基準を持った PBI になるのか、そのコツをつかむのに 1〜2 時間かかりました。その後、ワーキンググループは、残りの関連する優先度の高い PBI を、よく練られた PBI に改良するために、さらに 1〜2 時間の努力を必要としました。これらのワーキンググループの開発者は、通常、設立時に結成した BIOS 機能チームのうち 2 つ以上にまたがっていました。

BIOS コンポーネントバックログの詳細が十分であると感じたところで、ミーチャと私は、ストーリーマッピングと親和性推定に移りました。ミーチャと私は、各 PBI の要約詳細を小さなカード形式で印刷し、San Francisco BIOS チームが座っている場所の近くにある小さな情報ラジエーター専用ルームでストーリーマップの構築を開始しました。ミーチャと私は、BIOS エンジニアの時間的な制約に配慮しながら、意味のあるときにはいつでも、さまざまなグループの BIOS エンジニアを引き込みました。最終的には、新しい Intel チップセットとボード設計の MVP とともに、BIOS コンポーネントバックログの合理的な順序を作り上げました。MVP は、コンポーネントバックログの半分以上を占めることになりました。ストーリーマップが落ち着きを取り戻すまでには、数日間にわたり、それぞれ 1〜2 時間のセッションが必要でした。

この最初の改良作業の終わりに、私たちは BIOS チームのメンバー全員を集めて、壁に貼ってあるものを見直すために、より大きなミーティングを開きました。ポートランドチームのメンバーが効果的に参加できるように、デジタル写真とビデオ会議を組み合わせて使用しました。この大きな会議で、BIOS チームのメンバーは、ミーチャがストーリーマップの全体的な健全性をチェックするのを手伝いました。また、親和性推定を使用して、MVP 内のすべての PBI の大まかな工数推定値を得ました。

BIOS チーム全員の集合的な知識を最終的に把握したので、1〜2 スプリント先という短期的な視野に焦点を当てた、日常的な改良セッションに入りました。

LeSS の本からの様々なマルチサイト参照コンテンツは、以下の*マルチサイト参照コンテンツ*のサブセクションにリストアップされています。

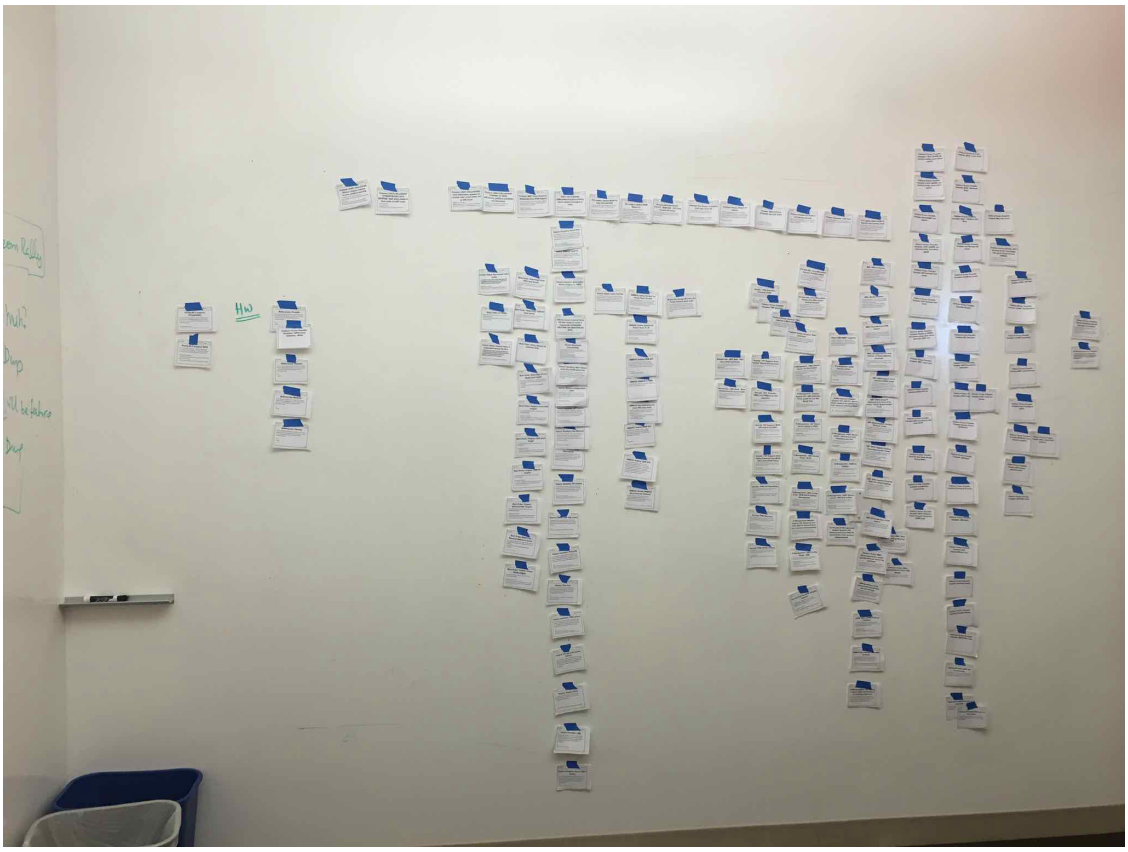

図 17: 最初の 2 日間の立ち上げ会議からの簡単な付箋紙の PBI は、さらに改良されて電子的に保存されました。これらの改良作業は、BIOS コンポーネントの特定の領域に焦点を当てた小さなクロスチーム・グループによって行われました。チーム横断のグループが収穫の少ない地点に到達するまでに、1 スプリント以上かかりました。チーム横断的な改善活動から十分な知見が得られたので、全体像を把握するために、物理的なフォーマットに戻しました。これは、ストーリーカードを印刷し、壁に無造作に貼り付けて、さらなる改善活動に備えているところです。

図 18：BIOS Fake Product Owner は、自然なグループ分けと順序付けを探し始めた。最終的な結果は、ストーリーマップと、壮大なグループ化を伴う蛇のように順序付けられたコンポーネントバックログを少し混ぜたようなものでした。この大きなマップは、数日間かけてゆっくりと進化していきました。BIOS の機能チームから様々なグループの人々が、より深い洞察を得るために引き抜かれました。壁が落ち着くと、偽のプロダクトオーナーは、BIOS スクラムチームのメンバー全員とミーティングを行い、全体的な健全性をチェックするようにしました。この時点で、赤い矢印で示すように、MVP が明らかになりました。

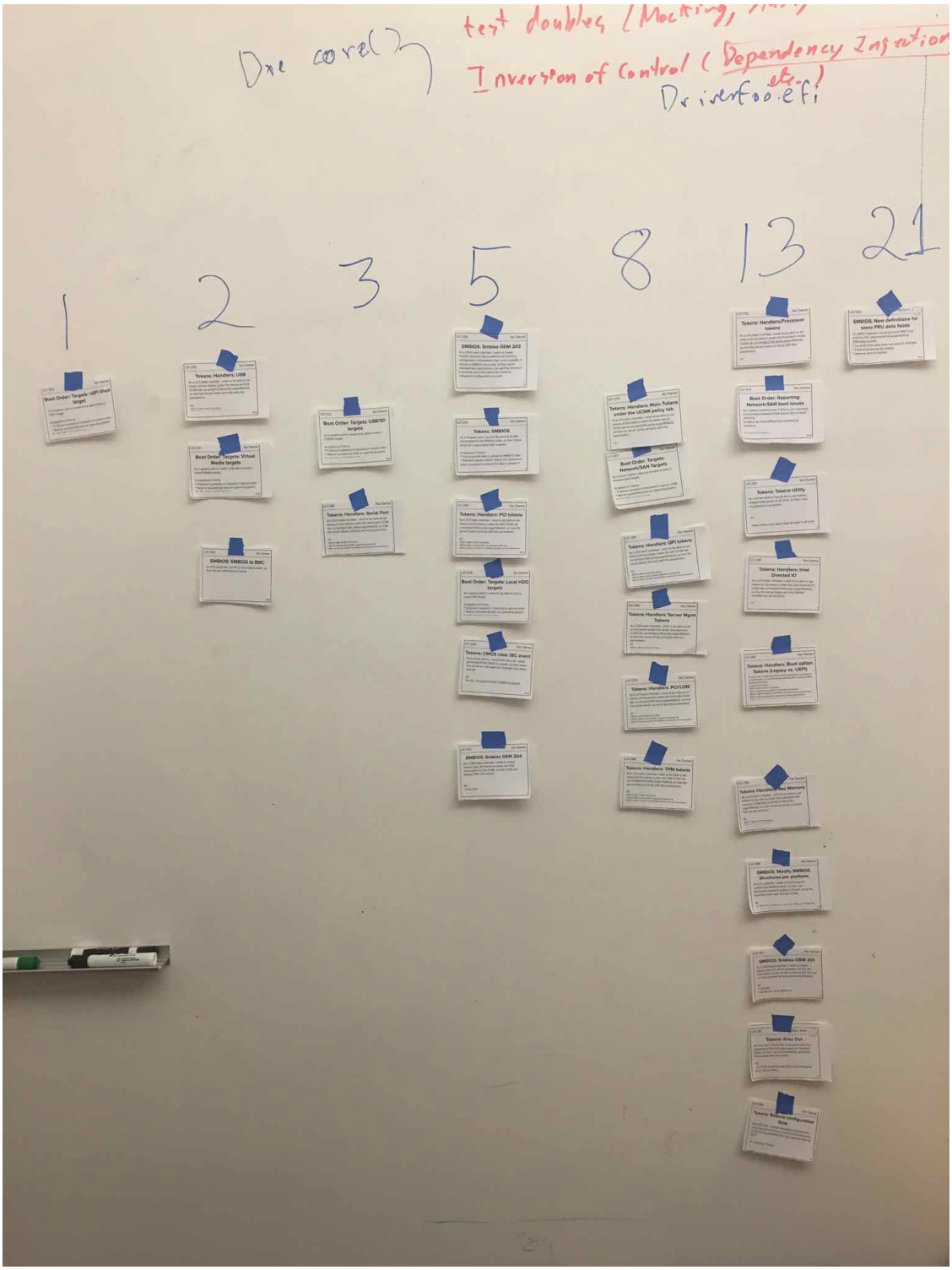

図 19: _大グループのマルチチームのサニティチェックの間、すべての BIOS スクラムチームメンバーが利用可能であったが、MVP 内の残りのすべての PBI を再推定するために親和性推定が使用された。その後、カードはよりきれいなストーリーマップのバージョンに再配置され、ラリーは新しい情報を反映するために更新されました。

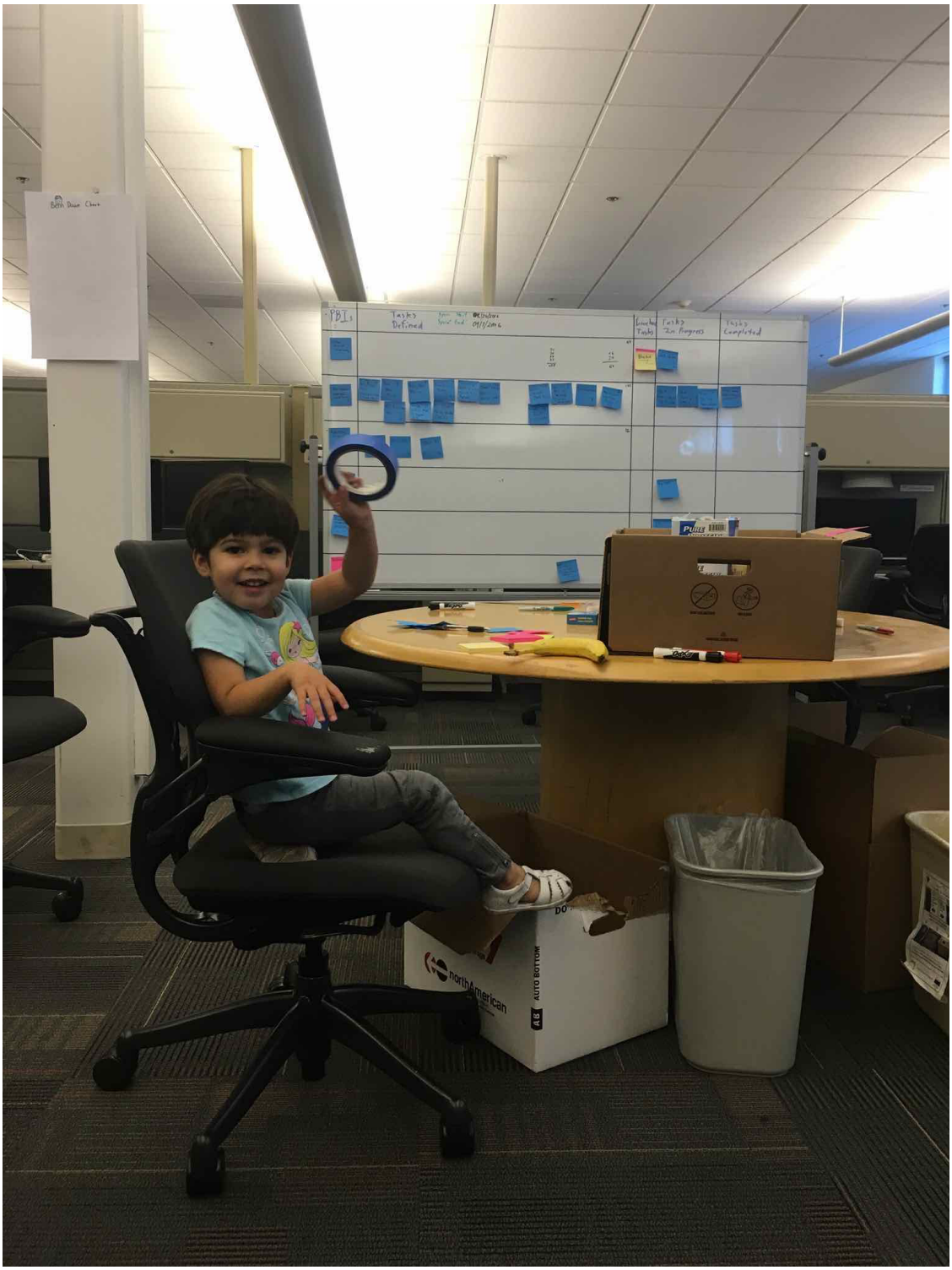

図 20：ミーチャは、私たちが必要とするマスキングテープを提供してくれるヘルパーがいることを確認しました。

マルチサイト参照コンテンツ

「Practices for Scaling Lean & Agile Development」には、多拠点会議の進行に関連する有用なコンテンツが多数掲載されています。大規模スクラムによる大規模、マルチサイト、オフショア製品開発_』に、マルチサイトミーティングのファシリテーションに関連する有用なコンテンツがたくさんあります。副題でさえも、それを暗示しています。

大規模スクラム More with LeSS には、関連する物語と、いくつかの関連するガイドも含まれています。

_Large-Scale Scrum: More with LeSS_のコンテンツは以下の通りです。

- _ガイド。クロスチーム・ミーティング
- ガイド マルチサイト PBR
- _LeSS Huge Story。マルチサイト・チーム

リーン&アジャイル開発のスケーリングのための実践」の第 12 章「マルチサイト」。大規模スクラムによる大規模、マルチサイト、オフショア製品開発_の第 12 章：マルチサイト_は、マルチサイトの課題への対処に完全に費やされている。この章の中で、最初の BIOS コンポーネントバックログの改良作業に最も直接関連するいくつかの実験があります。

- Try…Seeing is believing-ubiquitous cheap video technology and video culture（見ることは信じること-ユビキタスな安価なビデオ技術とビデオ文化
- 試してみよう…マルチサイト計画ポーカー（見積もりポーカー）
- Try…多拠点会議の基本プラクティス

初期のリファインメントを振り返って

最初の BIOS コンポーネントのバックログを改良するために費やした詳細と努力の量は、私が以前または以後に行った同様の作業の中で最も大きいことは確かです。しかし、このような状況であれば、もう一度やり直したいと思います。私たちは、BIOS エンジニアの整合性と参加意識を高めることができました。

将来、プロトタイプの Intel チップセットに対応するためにコンポーネントバックログを拡張する際に、作成した詳細な情報が必ず役に立ちます。AMI プラグイン アーキテクチャの利用が増えれば、手戻りが少なくなるため、詳細はさまざまになります。しかし、新しい PBI の種を蒔くには、以前に行われたものに目を通すのが一番でしょう。

私たちは、人工的な改良のタイムボックスに自分たちを押し込めるようなことはしませんでした。どのレベルまで明確にし、どのレベルまで詳細にするのが最も合理的かを判断し、それを達成するためにエネルギーを集中させたのです。そして、それが完了したら、今度はチーム横断のリファインメントミーティングを実施しました。

BIOS の Done の定義

2 つ目は、「Done の定義」の作成です。チームは、過去のひどいコピーペーストの行動から可能な限り離れることを確実にすることに決めました。可能な限り AMI のプラグイン可能な拡張ポイントを活用し、足りない必要な拡張ポイントは AMI に追加するよう圧力をかけることにしました。このコミットメントは、以下に含まれるより正式な Done の定義に表れています。以下の行をご覧ください。 「プラグインレイヤー以外の変更は行わない。

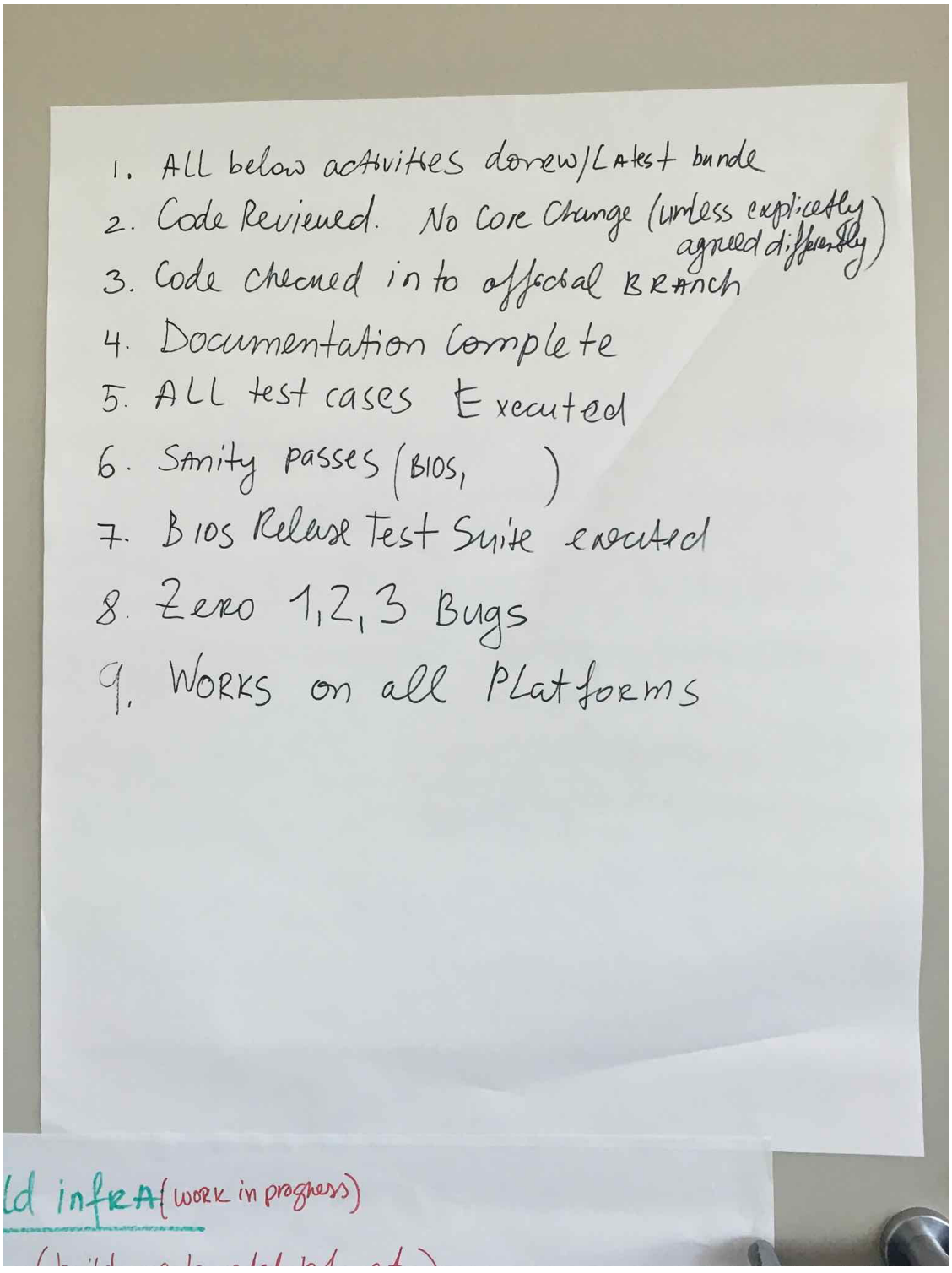

図 21：これは、数日間のローンチイベント中に BIOS チームによって作成された最初のドラフト Done の定義である。

Third Definition of Done (Firmware Component Teams in Waterfall Ecosystem)
Version: 1.23
Date: July 3, 2017

- ❑ All User Story acceptance criteria have been met.
- ❑ Relevant Epic acceptance criteria have been met.
- ❑ All activities below performed with latest third-party release bundle.
- ❑ Code peer reviewed in electronic peer review system.
- ❑ No changes outside of pluggable layer, unless agreed by third-party as an exception not handled by pluggable layer.
- ❑ Code checked into official branch.
- ❑ Documentation complete.
- ❑ Features documented via SW Spec Template and checked into document management system.
- ❑ All test cases executed (manual testing allowed).
- ❑ Test plan reviewed.
- ❑ Test management system used to track manual test cases.
- ❑ Zero defects in User Stories (within Dev Team control).
- ❑ Automated release test suite passes.
- ❑ Build is successful on all platforms and automated sanity test passes on all platforms.
- ❑ Legacy defects, or new defects from the waterfall side of the business follow the fire lane process.

図 22: BIOS チームが使用した Done の定義は、数回の Sprint の後、ここに示すように進化しました。この Definition of Done の例は、追加のコンテキストとともに、Forging Change の表 9.3 で見ることができる。

BIOS スクラムの主な役割

Trent、ミーチャ、そして私は、プロダクトオーナーとスクラムマスターの最適な選択について、非常に多くの考えを巡らせました。

詳細は後述しますが、結局、ミーチャがプロダクトオーナーとして唯一の賢明な選択となりました。

また、当初は私が全チームのスクラムマスターとして活動する必要があるという結論に達しました。他のスクラムマスターの選択肢の中には、必要な経験を持った人がいませんでした。もし、ミーチャをプロダクトオーナーにすることにこだわらなければ、ミーチャをスクラムマスターに選んでいたかもしれません。

私たちが選んだプロダクトオーナーとスクラムマスターについて、よりニュアンスの異なる洞察を以下に述べます。興味があれば読み、なければ読み飛ばしてください。

ミーチャ、一時的な偽プロダクトオーナーになる

ミーチャは、BIOS グループ全体のディレクターでした。ミーチャ、Trent、そして私は、プロダクトオーナーが開発チームのメンバーに対して直属の権限を持つことは、力関係が不均衡になるため、非常に問題があることを認識していました（下記参照）。別の人格がその役割を担っていた場合、これは重大な問題となった可能性があります。私たちの場合、ミーチャは生まれつき謙虚なサーバント・リーダーなので、この対立は大きな問題にはなりませんでした。

Trent、ミーチャ、そして私は、プロダクトオーナーとしてより良い選択をしようと考えました。最終的に私たちは、この役割を効果的に果たすのに十分な要素を持ち、かつ適切な性格の強さを持つ人が他にいないことに気づきました。他の候補者は、性格に問題があるか、MCS 製品全体と BIOS コンポーネントに関する十分な知識がなく、その役割を果たすことができなかったのです。

ラインマネージャーがプロダクトオーナーにならないようにする

私たちは、ミーチャをラインマネージャーと偽の一時的なプロダクトオーナーの両方として、私たちの特定の状況を回避できたからと言って、一般的に同じことを試みることをお勧めしません。LeSS の組織構造に関するガイダンスでは、チームメンバーがプロダクトオーナーに報告することを明確に避けています。

以下は、Large-Scale Scrum からの引用です。More with LeSS からの引用が参考になる。

> **ピンではなくピア**–チームがプロダクトオーナーに直接または間接的に報告し、階層的な力関係がある場合、その構造を変更し、チームとプロダクトオーナーがピアとして協力できるようにする必要があります。プロダクトオーナーは、チームをタスクのための小間使いのように扱うのではなく、協力的な関係を育むのです。5 つの関係性_**
>
> この組織構造で重要なポイントは、チームとプロダクトオーナーは「仲間」であり、上下関係にはならないということです。私たちは、役割の間で力のバランスを保つことが重要であると考えました。チームとプロダクトオーナーは、可能な限り最高の製品を一緒に作るために「協力的な」仲間であるべきで、ピア構造はこれをサポートします。この点については、「プロダクトオーナー」の章でさらに詳しく説明します。LeSS の組織構造

コンポーネントバウンダリのバランスに起因する PO の懸念事項

製品管理グループには、PO として迎えることを検討していた真のプロダクトオーナー候補がいました。残念ながら、関連する副社長の影響力の範囲に制限があったため、これらの個人のいずれかが管理するためのドメイン知識を持っていたマルチコンポーネントの境界になるように、コンポーネントの境界を十分に拡大することができませんでした。BIOS のマルチコンポーネント境界を拡張しても、MCS BIOS コンポーネントと MCS

製品全体の自然な複雑さは、一般的な企業情報システムよりも根本的に大きいことを覚えておくことが重要である。

我々は、部門内の最も上級の幹部の一部から賛同を得ていたが、それらの幹部は、対処するために他の競合する多くの懸念を持っていた。しかし、そのような状況下でも、私たちの取り組みに政治的な圧力をかけてくれた幹部がいたことは間違いない。しかし、現実には、私たちに代わってさらに大きな組織改革を行うための政治資金を得るには、時間がかかり、さらに大きな成功を収める必要がありました。

高度に専門化した BIOS エンジニアが、さまざまな BIOS サブコンポーネントを横断して作業することに慣れるまでは、BIOS マルチコンポーネントの境界をさらに拡大すると、BIOS チームの技術力とチーム力が持続可能なレベルよりもさらに急速に伸びることになります。したがって、早い段階で BIOS コンポーネントの境界をさらに拡大することは、いずれにせよ選択肢とはならなかったのです。これらの理由は、LeSS ガイダンスが製品定義を段階的に拡大し、LeSS Huge 採用において要件領域を段階的に追加する動機付けとなるものです。

関連する LeSS 製品規則

特に、以下の厳選された LeSS フレームワークのルールは、関連性が高い。

- 出荷可能な完全な製品に対して、1 人のプロダクトオーナーと 1 つのプロダクトバックログが存在する。
- プロダクトオーナーは単独でプロダクトバックログの洗練に取り組むのではなく、顧客/ユーザーやその他のステークホルダーと直接作業する複数のチームによってサポートされる。
- 優先順位付けはすべてプロダクトオーナーを通すが、明確化はできるだけチームと顧客/ユーザーやその他のステークホルダーとの間で直接行う。

プロダクトオーナーシップは最大の課題ではない

拡張 BIOS マルチコンポーネントの観点から見ると、顧客の問題はむしろ単純である。各新しい Intel チップセット用のカスタム BIOS は、前世代の Intel チップセット上で動作する前世代の BIOS が提供するのと同じサービスを提供しなければならない。これらの機能のマイナーな拡張は、新しいチップセットの世代ごとに必要とされることが多いが、それでも基本機能に比べれば些細なことである。BIOS の Fake Product Owner は、ある程度、将来的に問題を解決するためのより良い方法をチームと一緒に考える専門家に過ぎなかったのである。

コードベースの貧弱な状態は、「契約ゲーム」によって推進された歴史的に劣悪なエンジニアリング慣行の結果であった。非現実的な人工的な「コード・コンプリート」の期限を守るよう、常に上からプレッシャーをかけられ、職人気質のための個人の安全が十分に確保されていなかったのです。大量のコピーペーストの再利用と自動テストカバレッジの低さは、「コントラクトゲーム」がもたらした自業自得の 2 大例と言えるでしょ

う。その結果、BIOS コードベースを新世代の Intel チップセットに適合させるという途方もない作業が人為的に行われることになったのです。

皮肉なことに、より包括的な「Done の定義」と、すべてのテストを段階的に完了させることのできる部門横断的なチームがあれば、より良い仕事をするための十分な時間が常にあったのである。インテルのチップセットのリリース日は本当だったが、ウォーターフォールステージのゲートデイトは、ウォーターフォールプロセスの押し付けによって生じた人工物だったのである。

新しい BIOS チームによって生み出された劇的な品質向上が、この状況を変えていくことになるはずだった。BIOS のカスタマイズの大部分がプラグイン可能なレイヤーになり、より優れた自動テストが行われるようになれば、新しい Intel チップセットへの適応も、従来よりもはるかに少ない労力で可能になると思われます。さらに、改善努力を続けることで、BIOS チームは、MCS Admin レイヤーを通して作業する能力を完全に拡大することができた。

品質向上とコンポーネント境界の拡張の両方が十分に進んだら、新しいチームはよりダイナミックな要件を処理するための時間とスキルの両方を手に入れることができます。それまでは、拡張された BIOS のマルチコンポーネントの目標は非常に明確で簡潔であり、関連する作業も膨大だったため、プロダクトオーナーの役割はあまり必要なかったのです。実際、長年の BIOS 経験を持つこの BIOS エンジニアのグループは、すでに何度も実装してきたものをより良く実装する方法について、新しい道を切り開く必要があっただけなのです。

BIOS チームがまとまり始め、エンジニアリング手法が改善され、BIOS のマルチコンポーネント境界が MCS 製品全体の自然な要件領域を形成する MCSA まで拡張されると、製品管理からの真のプロダクトオーナーがいずれ重要になるだろう。

プロダクトオーナー選定制約の概要

これらの理由を総合して、私たちはミーチャをプロダクトオーナーの中期的なベストチョイスとして選びました。つまり、私たちは「リーン開発とアジャイル開発のスケールアップのための実践」で説明されている「偽プロダクトオーナー」の実験と、「大規模スクラム」の「一時的な偽プロダクトオーナーによる早期開始または混乱」のガイドを実施することを選んだということです。『*More with LeSS*』の *Start Early or Messenger with Temporary Product Owner* のガイドを参照してください。

- BIOS チームの過剰な専門性と局所的な焦点に加え、長年の貧弱なエンジニアリングプラクティスによってもたらされたさらなる複雑性が、偽のプロダクトオーナーを開始することを要求しました。この人物は、BIOS コンポーネントバックログを理解し、BIOS チームの尊敬を維持するのに十分な専門的技術的深みを持つ人物である。

- 元来、他人に力を与える傾向があり、マイクロマネジメントを避けるプロダクトオーナーは、BIOS チームが成長し成熟できるような支援的な状況を確立するのに重要な存在だったのです。
- BIOS の LeSS 指向の構造は、依然として契約ゲームによって駆動される大規模なウォーターフォールのコンテキスト内で実行されていたので、BIOS チームのための支援コンテキストを確立するのに役立つ十分な地位と政治的影響力を持つプロダクト オーナーを持つことが重要でした。
- コンポーネント境界を拡大することで、BIOS コンポーネントバックログを理解し導くために必要な BIOS 固有の知識を減らすことができ、それによってプロダクトオーナーの他の良い選択が可能になる。BIOS エンジニアの機能横断性が高まり、これが可能になるまでは、BIOS の知識を多く持つ Fake Product Owner が必要であった。
- Fake Product Owner の存在は、BIOS チームを MCS 製品全体の顧客からさらに孤立させることになる。私たちは、できるだけ早く Fake Product Owner の必要性をなくしたいと考えています。

スクラムマスターとしてのコーチ

MCS 部門には、BIOS チームで試みるような厳しい状況下で、Scrum チームの設立を効果的に指導できるだけの経験を持つ正社員はいないと判断していた。選ばれた正社員は、徐々にその役割を指導していく必要があります。

Trent と私は、ミーチャが優れたスクラムマスターになるための適切な性格、人間関係、組織的知識、コンポーネント知識を持っていると感じていました。しかし残念ながら、コンポーネントの境界をさらに拡大するまでは、ミーチャにプロダクトオーナーの役割を担ってもらう必要がありました。

私たちは最終的に、最初は私がスクラムマスターとして活動する必要があると判断しました。Trent と私は、私がその役割を果たすための帯域幅を確保することに気を配りました。私は、BIOS エンジニアリングエリアの空いているデスクに移り、その後数ヶ月は、BIOS チームの立ち上げと成熟に専念しました。毎週、ポートランドに通っていたので、いつでも都合のいいときに移動することができました。実際、ミーチャと私は、ほとんどのことを一緒にやっていました。私は、ミーチャや他の強力なナチュラル・リーダーが、ほとんどの活動を自らリードできるよう、継続的にサポートしました。

BIOS チームがまとまり、成熟し始めると、さらに何人かの自然なスクラムマスター候補が現れました。彼らはもともと必ずしもスクラムに取り組んでいたわけではなかったのですが、時間の経過とともにそうなっていきました。ミーチャと私は、このような特定の人たちを、将来スクラムマスターになる可能性のある人たちに成長させることに焦点を当てました。

ミーチャの次の推薦文は、その困難な状況を明らかにしている。

> James との仕事は、私にとっても、チームにとっても変革的な経験でした。
>
> 彼は、私たちを「ここでスクラムができるわけがない」から「もう後戻りできない」状態にしてくれました。彼は、チームをウォーターフォールからスクラムへと導いたのです。
>
> 私たちはファームウェアエンジニアのチームであり、スクラムは私たちの分野にはふさわしくないと固く信じていました。ジェームズは、スクラムがソフトウェアクラフトマンシップ、品質、そして仕事の計画の仕方に焦点を当てるためのフレームワークであることを実証したのです。
>
> James は、私がプロダクトオーナーとして行動できるように指導し、チームがスクラムフレームワークの学習曲線を乗り越えるように指導してくれました。
>
> 開発、テスト、CI/CD の技術的な側面に焦点を当てた彼の指導は、チームにとって非常に有益なものでした。彼は多くのコンセプトやツールを紹介し、必要な技術面ではチームメンバーと協力してくれました。
>
> 結果は非常にわかりやすいものでした。私たちはスクラムを 1 年近く実践しています。バックログの可視性が格段に向上しました。チームワークが良くなった。初期の製品品質が向上した。

スクラムマスター選択制約の概要

- 異国情緒あふれるハードウェアとファームウェアの技術的背景、そして文化的な課題は、フルタイムのスクラムマスター候補者のスキルセットには難しすぎるものでした。
- ミーチャは、効果的なスクラムマスターに短期間で成長する素質がありました。このオプションは、ミーチャが偽のプロダクトオーナーとして行動する必要があったため、除外されました。
- コンポーネント境界を拡張することが政治的にも技術的にも現実的になれば、ミーチャをプロダクトオーナーに置き換え、ミーチャや有望で成長を見せた人物をスクラムマスターの役割に移行させることができます。
- 私は、このような難しいスクラムの採用を試みるのに十分な、強力なスクラムマスターのスキルセットを持つ唯一の人間だったのです。私は、BIOS 開発者やミーチャなどのガイダンスに大きく依存しながら、集団で物事を改善する方法を見つけることでしか、それを行うことができませんでした。

BIOS Teams 自ら選ぶ

インセプションでは、それぞれの新しいチームに誰が入るかを共同で考えました。

ポートランドの人たちが 1 つのチームを作ることは、グループにとって簡単な決断でした。ポートランドには、6 人ほどの人しかいませんでした。その中には、ソフトウェア、

ハードウェア、テストのエンジニアも含まれていた。ミーチャ、クリシュナ、トレント、そして私の 4 人は、ポートランドでトレーニングする人を決めたときから、そうなるように努力してきました。クリシュナは、MCS の他の部分を担当しているグループから引き抜いたテストエンジニアの中から、適切な人材を見つける手助けをしてくれたかもしれません。

本当の課題は、誰がサンフランシスコのどのチームに入るのか、そしてサンフランシスコのチームがいくつあるのか、ということでした。当初は 2 名でスタートしました。確か、最初の打ち上げの日の午後に、チーム編成を自分で考えてみたんです。その後、一晩考えて、2 日目にはチーム編成を少し修正しました。

数カ月後には、BIOS のコンポーネント境界の端で作業する開発者を数人加え、同じ LeSS 指向の構造で作業する別のチームを 1 つまたは 2 つ形成する予定です。

ポートランドのチームは、ほぼ同じ場所で活動していましたが、当初は重要なエンジニアリングスキルがいくつか不足していました。しばらくは、サンフランシスコのエンジニアが 1〜2 人、ポートランドチームのメンバーとして参加していました。やがて、ポートランドのエンジニアは完全に自立していったと思います。

BIOS のケーデンスとスプリントタイミング

そこで、2 週間の周期で行うことにしました。スプリントの境界線が週の半ばになるようにしたのは、私が成層圏のどこかで飛行機に乗っているのではなく、チームのために使えるようにするためです。

BIOS レトロスペクティブの構造

私たちは、チーム横断的なレトロスペクティブとチームごとのレトロスペクティブを組み合わせて、いくつかのバリエーションを試しました。レトロスペクティブの構造は、細部では若干異なるものの、精神的には LeSS スプリントルールに類似していた。

レトロスペクティブのメタ構造の観点から、私たちは通常、Esther Derby と Diana Larsen による_Agile Retrospectives_に見られるアドバイスに従いました。最も一般的なテクニックは、ドット投票による単純なブレーンストーミング、タイムボックスによる議論、そして回顧構造そのものに関する簡単な 4 箱の回顧の使用であった。

初期のレトロスペクティブの構造

当初のレトロスペクティブな構成は以下の通りであった。

1. チーム横断ミーティングを行い、チーム横断的に最も優先度の高い議論項目を特定する
2. チームごとのレトロスペクティブで、チームごとの問題点とチーム横断の最優先事項を議論する
3. チームごとの会議の結果を議論し、チーム横断的な問題を解決するためのアプローチについて合意するためのチーム横断会議（LeSS 全体レトロスペクティブ）。

クロスチーム・ミーティングには、いずれも BIOS 各チームのメンバー全員と、偽プロダクトオーナーのミーチャ、そしてスクラムマスター代行の私が参加しました。

最初のクロスチーム・ミーティングは、チームレベルのレトロスペクティブで検討することが重要だとグループが感じた、チーム横断的な課題を特定することに焦点を当てました。その意図は、チーム横断的な問題に対する潜在的な解決策を、より小さなチームレベルのディスカッションで考え出す機会を提供し、その後のクロスチーム・ミーティングでは、より大きなグループのディスカッションでは表面化しなかったかもしれない深い洞察から恩恵を受けるようにすることでした。

これは、チームレベルのレトロスペクティブの後に全体のレトロスペクティブを行うという LeSS フレームワークの構造に非常によく似ていることにお気づきでしょう。重要な違いは、チームレベルのレトロスペクティブは、全体的なレトロスペクティブの第 1 部と第 2 部の間に挟まれていることです。

実際には、チームレベルのレトロスペクティブは、全体的なレトロスペクティブの最初の部分で特定されたあらゆる問題にほとんどの時間を費やしていたようです。もし、LeSS フレームワークが規定するように、チームレベルのレトロスペクティブが全体的なレトロスペクティブ会議の前に実施されていれば、チームレベルのレトロスペクティブはより深い洞察に満ちたものになっていただろうと思います。

表面化した問題のほとんどは、チーム横断的なものでした。これは単に文脈の結果なのか、それともチームレベルのレトロスペクティブの前に全体のレトロスペクティブの一部を行うことで、意図せずチームレベルの議論からチームの注意をそらす効果があったのかは分かりませんが、チームレベルのレトロスペクティブはより困難なものでした。

後期のレトロスペクティブの構造

私たちは一般的に、チーム横断の最終回顧展を、回顧展そのものの回顧展で締めくくるように気をつけていました。このことから、最終的には以下のようなレトロスペクティブの構成に変更することにした。

1. チーム横断会議による、チーム横断の最優先検討事項の洗い出し
2. 上位数点の課題についての分科会。2. チームの垣根を越えて、自分が最も熱意をもって取り組めるもの、あるいは最も必要とされているものに参加する。
3. チーム横断会議：分科会の成果を議論し、チーム横断的に議論された課題を解決するためのアプローチについてコンセンサスを得ます。

この構成は、真ん中のステップを除いて、最初の構成と同じであることがおわかりいただけると思います。チームごとのレトロスペクティブに分かれるのではなく、チームの垣根を越えて、最も親和性が高いと思われるグループと連携するワーキンググループに分かれたのです。

実際には、ポートランドチームは第 2 ステップの間、通常チームとして活動しました。チーム横断的なイベントは、テレプレゼンスルームで行うのが一般的でした。もし、サンフランシスコの誰かが、ポートランドのメンバーの大半が参加したトピックに特に熱意を感じていたら、そのトピックのためにテレプレゼンスルームを使用することになります。他のワーキンググループは、テレプレゼンス機能が限られた他のブレイクアウトルームを使用することになります。

サンフランシスコでの振り返りワーキンググループの分かれ方は、ややランダムなことが多かったですね。会議テーブルの真ん中に分かれて、数人がワーキンググループを交代することがよくありました。

関係者の生来の協調性と人間関係の歴史を考えると、チームごとの専用の回顧がないことは問題ではないように思えました。私が関わってきた他のグループでは、チームごとのレトロスペクティブが重要であることがよくわかりました。おそらく、時間が経ち、チームが増えれば、力学が変化し、チームはチームレベルのレトロスペクティブを復活させることを決めたことでしょう。

オーバーオールレトロスペクティブでは対処しきれない、より大きな構造的な問題

LeSS では、全体的な回顧は、より広範なシステム的問題に対処することを目的としている。BIOS チームは、自分たちの管理下にあるチーム横断的なシステム上の問題には積極的に取り組み、解決策を実行したが、より広範なシステム上の問題にはほとんど取り組むことができなかった。VP 層マネージャーも、新しい SVP マネージャーも、一度も出席していない。

ガイドを参照。大規模スクラムの_全体的な振り返り_を参照してください。このトピックに関するより詳しいニュアンスについては、_大規模スクラム: LeSS でさらに詳しく_を参照してください。

BIOS と LeSS ルールの整合性

BIOS チームからの詳細や洞察を述べる前に、正式な LeSS ルールとの整合性を確認することが有益である。

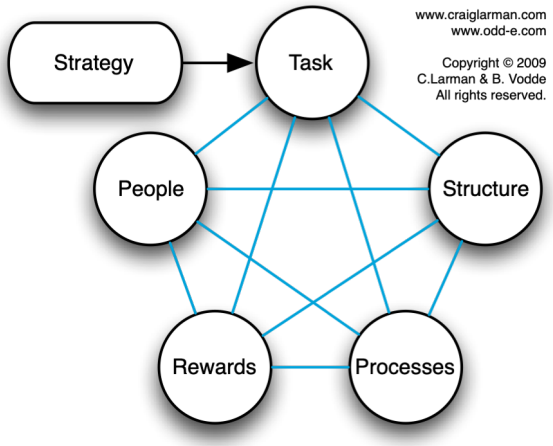

図 23：Scaling Lean and Agile Development の中で、Larman と Vodde は Jay Gilbrith の星型モデルの観点から組織戦略について述べている。LeSS の採用における最大の問題のいくつかは、報酬と人材、そして、それらと構造との交差に関連している。技術的熟練と管理的熟練を同じレベルで報いる報酬構造への十分な配慮と相まって、正式な組織構造をフラット化することに失敗したのである。BIOS の LeSS ルールへの整合](#bios-alignment-to-less-rules)のセクションを読むときに、ギルブリスのモデルを心に留めておくと便利です。（図は許可を得て使用しています。）

LeSS 構造の枠組み

- 実際のチームを基本的な組織の構成要素として、組織を構築する。

  はい

- 各チームは、（1）自己管理、（2）機能横断的、（3）同じ場所にいて、（4）長寿命である。

  セルフマネジメント： すべてのチームは、初日からセルフマネジメントを行いました。各チームはスプリントにどれだけの仕事を取り込むかをコントロールし、各スプリントを共同でタスク化し、BIOS コードベース内のアーキテクチャの決定を完全にコントロールし、毎日のスクラムを実行し、その他セルフマネジメントのチームに期待されることすべてを実行したのです。

  クロスファンクショナル： 初期には、各チームが BIOS コンポーネントの特定の領域に強い親和性を持っている傾向がありました。BIOS コンポーネント内でもよりクロスファンクショナルになるには、各チームともしばらく

時間がかかりました。チームがさらに進化するにつれ、当初の親和性から徐々に外れていくようになったのです。

共通の BIOS コンポーネントバックログと単一の Fake Product Owner の存在は、チーム内およびチーム間のクロストレーニングのレベルを継続的に高める構造的な力を生み出した。

BIOS チームが MCSA を通じて BIOS コンポーネントの境界をさらに拡張するために働くと、チームは徐々に MCS 製品全体の観点からさらにクロスファンクショナルになっていきました。これは完全な可能性に達することはありませんでしたが、MCSA 関連の BIOS 作業に親和性のあるチームは、確実にその道を歩んでいました。このトピックは、_*BIOS コンポーネント境界の拡張*_ のセクションでさらに検討されます。

各チームはほとんど同じ場所に配置されました。 ポートランドチームは BIOS コードベースのいくつかの分野で知識が不足していたため、ポートランドチームは当初、サンフランシスコ地域に 1〜2 名のメンバーを配置していました。このような例外的なケースを解決するために、各チームは積極的にクロストレーニングに取り組みました。確か、サンフランシスコを拠点とするポートランドチームのメンバーは、数回のスプリントでポートランドチームを離れ、サンフランシスコチームのいずれかに参加することができたと記憶しています。

長続きしたこと チームは数ヶ月間、ほぼそのままでした。大規模なレイオフは、明らかに長寿命化に影響しました。また、ポートランドチームの分散が解消され、開発中のハードウェア世代の BIOS に対応できる BIOS 開発者が増えたため、チーム間でメンバーの自主的な入れ替えが少しありました。一般的には、新たに加わったメンバーが新しいチームを結成し、元の 3 つのチームはかなり安定した状態を保っていました。

しかし、数カ月ごとに新しいチームに移籍するようなことはありませんでした。

数年のタイムスケールから長寿を見ると、確かに問題がありました。Larman 氏と Vodde 氏は、「Scaling Lean and Agile Development」の中で、組織戦略について Jay Gilbrith 氏のスターモデルという観点から語っている。
様々な要素のうち、「報酬」と「人」の要素には十分な注意が払われていない。

既存の報酬体系では、エンジニアが技術的な道を歩み続けるインセンティブはすでにある程度備わっていたものの、管理職として出世するための金銭的・文化的インセンティブがはるかに強く残っていたのです。同様に、部門の最高レベルで行われているコミットメントゲームから生じる継続的な圧力釜は、重要な学習や専門的な成長のための時間をほとんど残さないままであった。

- チームの大半は、顧客志向のフィーチャーチームです。

  各チームは BIOS コンポーネントの境界でほぼ垂直方向にスライスされていましたが、BIOS コンポーネントの境界はそれ自体、妥協の産物でした。BIOS チームが成熟するにつれ、私たちは MCS 製品全体を貫く、より垂直方向に拡張されたマルチコンポーネント境界を確立しようと努力しましたが、最終的に達成できたかもしれないほど垂直方向にはなりませんでした。

- スクラムマスターは、LeSS の導入がうまく機能するように責任を持ちます。彼らの焦点は、チーム、プロダクトオーナー、組織、および開発プラクティスに向けられます。

  はい

- スクラムマスターは、1 つのチームだけでなく、組織全体のシステムに焦点を当てます。

  スクラムマスターは、専任のフルタイムの役割です。

  1 人のスクラムマスターが 1～3 チームを担当することができます。

  米国にいるすべての BIOS エンジニアがフィーチャーチームに組み込まれる頃には、私は 5 つのチームにサービスを提供していました。実際には、ミーチャをはじめ、スクラムマスターに育てつつある人たちと多くの負担を分かち合っていました。

- LeSS では、管理者はオプションですが、管理者が存在する場合、その役割は変化する可能性が高いです。彼らの焦点は、日々の製品作業の管理から、製品開発システムの価値提供能力を向上させることに移ります。

  そうですね。ミーチャ、Trent、私の 3 人で、非常にフラットな体制を維持しています。ミーチャが日々の BIOS コンポーネント作業の管理に注力した記憶はありません。

- マネージャーの役割は、Go See の実践、Stop & Fix の奨励、“適合性より実験”など、製品開発体制を改善することです。

  ミヤと私は、SFO を拠点とするチームに同席していましたが、時々ポートランドにも飛んでいました。Go See と experiment over conformance の様々な例を挙げます。

  - Dmitry と私は、有益と判断された場合には、1 人以上の BIOS 開発者とペアでプログラムしたり、設計の議論に参加したりすることがありました。

  - Dmitry と私は、組織の上下左右にいる様々な人たちと積極的に会い、多くの組織的な障害を解決する手助けをしました。

  - Emmett Brown とペアを組み、BIOS コードの自動ユニットテストを作成する方法を研究しました。

  - BIOS チームがコントロールできる CI サーバーを立ち上げるために、数人の開発者とペアを組み、多くの時間を費やしました。

  - BIOS チームの振り返りでは、Sprint ごとに少なくとも 1 つの実験を特定し、その後のそれぞれの Sprint でアクションを起こしました。
- 製品グループについては、LeSS の完全な構造を「最初に」確立すること、これが LeSS 採用のために不可欠です。

  はい、拡張 BIOS コンポーネントの範囲内です。いいえ、より広い全体的な MCS 製品の範囲内です。
- 製品群を超えた大きな組織では、Go と See を使って進化的に LeSS を採用し、実験と改善が当たり前の組織を作りましょう。

  はい

LeSS プロダクトの枠組み

- 出荷可能な完全な製品に対して、1 人のプロダクトオーナーと 1 つのプロダクトバックログが存在します。

  BIOS コンポーネントについては Yes。MCS 製品全体ではありません。
- プロダクトオーナーは単独でプロダクトバックログの洗練に取り組むのではなく、顧客/ユーザーやその他のステークホルダーと直接作業する複数のチームによってサポートされる必要があります。

  はい

- 優先順位付けはすべてプロダクトオーナーを通すが、明確化はできるだけチームと顧客・ユーザーなどのステークホルダーとの間で直接行う。

  BIOS コンポーネントについては Yes。MCS 製品全体ではありません。

- 製品の定義は、実用的な範囲で、エンドユーザーや顧客を中心に幅広く設定する必要があります。時間の経過とともに、製品の定義は拡大する可能性がある。より広範な定義が望まれる。

  BIOS コンポーネントの境界を可能な限り広く拡張するという点では Yes。
  MCS 製品全体を包含する LeSS コンテキストの欠如という点で、「いいえ」。

- 製品全体の Done の定義は 1 つで、全チームに共通です。各チームは、共通の定義を拡張することで、より強力な Done の定義を持つことができる。

  BIOS チームによって使用される共通の「完了定義」の点では Yes。MCS 製品全体から見ると、「いいえ」。

- 完成目標は、Done の定義を改善し、Sprint ごとに（あるいはもっと頻繁に）出荷可能な製品に仕上げることである。

  BIOS コンポーネントに関してはできていましたが、MCS 製品全体に LeSS の採用を拡大するまでは、避けられない Undone の作業が少しありました…。

LeSS スプリントの枠組み

- 製品レベルのスプリントは 1 つであり、チームごとに異なるスプリントではありません。

  BIOS のコンポーネントチームという点ではそうですね。

- 各チームは同時にスプリントを開始し、終了する。各スプリントの結果、製品全体が統合されます。

  BIOS のコンポーネントチームという点ではそうですね。

- スプリントプランニングは 2 つの部分から構成されています。スプリントプランニング 1 は全チーム共通で行い、スプリントプランニング 2 は通常各チームで別々に行います。スプリントプランニング 2 は、密接に関連する項目については、共有スペースで複数チームのスプリントプランニングを行う。

  BIOS のコンポーネントチームという点ではそうですね。

- スプリントプランニング 1 には、プロダクトオーナーとチームまたはチームの代表者が参加します。各チームがそのスプリントに取り組む項目を仮選定します。各チームが協力する機会を確認し、最終的な質問を明確にします。

  BIOS のコンポーネントチームという点ではそうですね。

- 各チームはそれぞれのスプリントバックログを持っています。

  BIOS のコンポーネントチームという点ではそうですね。

- スプリントプランニング 2 は、チームが選択したアイテムをどのように実行するかを決定するためのものです。これは通常、設計とスプリントバックログの作成に関係します。

  BIOS のコンポーネントチームという点ではそうですね。
- デイリースクラムはチーム毎に行われている

  BIOS のコンポーネントチームという点ではそうですね。
- チーム間の調整はチームが決める。中央集権的な調整よりも、分散的で非公式な調整を優先する。コードでのコミュニケーション、チーム横断ミーティング、メンター、トラベラー、スカウト、オープンスペースなどによる Just Talk やインフォーマルなネットワークを重視する。

  BIOS のコンポーネントチームという点ではそうですね。
- プロダクトバックログリファインメント（PBR）は、共有学習を増やし、調整の機会を活用するために、複数のチームで行うことが望ましい。

  BIOS のコンポーネントチームという点ではそうですね。
- プロダクトのスプリントレビューは 1 つで、全チームに共通です。適切なステークホルダーが参加し、効果的な検査と適応に必要な情報を提供できるようにすること。

  BIOS のコンポーネントチームという点ではそうですね。
- チーム毎にスプリントレトロスペクティブが実施されている

  当初は BIOS のコンポーネントチームという観点ではそうでした。その後、先の「*BIOS Retrospective Structure Adaptation*」の項で説明したように、少し逸脱しました。
- チームレトロスペクティブの後に全体レトロスペクティブを開催し、チームやシステム全体の問題を議論し、改善策を練る。プロダクトオーナー、スクラムマスター、チーム代表者、マネージャー（いる場合）が参加します。

  BIOS のコンポーネントチームという点ではそうですね。

BIOS トリアージルール

BIOS チームは、MCS システム全体の他の部分に取り組んでいるエンジニアや経営陣からの要求に気を取られることがよくありました。これらの要求の中には、MCS 開発全体のために迅速な対応が必要なものもあれば、迅速に拒否するか、BIOS コンポーネントバックログに移動して優先順位をつけるのが最善と思われるものもあった。

一部のテストエンジニアを除き、BIOS スクラム開発チームのメンバーは全員、ミーチャ に直接報告した。機能管理者が、彼らに報告する BIOS チームメンバーの気を散らすという問題は、ほとんど存在しなかった。

この問題は、実際の MCS 製品の境界線と比較した場合、BIOS コンポーネントの境界線が不完全であることと、MCS BIOS コードと AMI BIOS コードが過剰に結合していることが主な要因でした。

トリアージガイドラインの正式なセットを作成すると、根本的な根本原因がより完全に対処できるようになるまで、これらの混乱を管理する上で非常に有用であった。以下のような利点がありました。

- プロダクトオーナーの意思の尊重
- 予期せぬ重要な要求に対応することができる
- BIOS エンジニアが不適切な要求に対してノーと言えるよう支援する
- 関係者全員への透明性の提供
- BIOS エンジニアの集合的な知恵の集約
- BIOS チーム内の整合性の達成
- BIOS スクラムチームが過去のデータに基づいてよりよく検証し、適応する能力

Example Triage Guidelines
Version: 0.33
Date: July 4, 2017

Add to the **Sprint Backlog**, and start work immediately if:

- ❑ I see an email from partner company regarding future platform with subject [Blah-blah] or [Blah-RC]
- ❑ I see an email from build system team, build is broken
- ❑ I see an email indicating the automated long-duration system test failed
- ❑ I see an email from the partner company with Future Platform label notification
- ❑ Platform team believes delaying the work will affect hardware schedule
- ❑ The issue is known to be blocking manufacturing

Add to the **Sprint Backlog**, and start work after approval (PO will likely say yes) if:

- ❑ I get an email from partner company regarding a shipping platform with the subject [Blah-blah] or [Blah-RC] and I then convince myself this is a critical release that needs immediate attention and cannot wait until the next Sprint
- ❑ Severity 1 or Severity 2 bug
- ❑ Issue is raised by manufacturing, but not a blocking item
- ❑ Issue is raised by HW/[Sub-System X]/[Sub-System Y]/[Sub-System Z] teams, but not a blocking item

Add to the **Product Backlog**, we'll prioritize during Product Backlog refinement meetings if:

- ❑ I get an email from partner company with Shipping Platform label notification
- ❑ I get an email with "+Bob" without clear indication of urgency
- ❑ Someone says: "Hi, we need Sam or Sally's help with …"
- ❑ New Severity 3 or lower bug is reported.
- ❑ Asked to work on old Severity 3 or lower bug
- ❑ Platform issue reported, but is not bound to an immediate date

図 24：BIOS チームが使用するこのトリアージガイドラインは、3 つの基本カテゴリーを確立している：即時行動を起こす、プロダクトオーナーに尋ねる、BIOS コンポーネントバックログに載せる、である。プロダクトオーナーは意図を明確に伝えると同時に、スクラム開発チームが適切なときに即座に行動を起こせるように権限を与えている。ガイドラインを明示することで、プロダクトオーナーは、チームの知恵を結集してガイドラインを共同で改良する確率も向上させることができます_。

BIOS のユニットテストの可能性

意思あるところに、方法あり。BIOS 内の自動ユニットテストは非常に困難であることは分かっていました。開発の初期段階では TCP/IP ネットワークスタックが存在せず、標準的な C ライブラリ関数もほとんどなく、すべてのコードをクロスコンパイルしてターゲットにフラッシュしなければならず、AMI BIOS エコシステム内のユニットテストに対する既存のアプローチも見つかりませんでした。

私はソフトウェアエンジニアとして 10 年以上、テストファーストの開発を実践してきましたが、BIOS や低レベルのハードウェア設計についてはほとんど何も知りませんで

した。同様に、BIOS エンジニアは、よりパフォーマンスの高いミドルウェアチームに見られる言語、ツール、デザインパターン、様々な職人技の実践にほとんど触れることがなかったのです。多くの点で、MCS の BIOS エンジニアのほとんどは、非常に狭いソフトウェアエンジニアリングのスキルを持つ電子工学のウィザードです。

Emmett Brown は情熱的なエンジニアで、MCS BIOS コードのテストファースト開発へ移行する方法を見出そうと決意していました。私は Emmett と一緒に、必要な設計要素を理解し、さまざまな上位の開発エコシステムにおけるそれぞれの事例を紹介しました。Emmett は私に、UEFI 規格や BIOS ツールチェーン、関連するハードウェアの制約など、さまざまな詳細を教えてくれました。そして、私たちは課題を解決するために、さまざまなアプローチを検討しました。

数週間の作業を経て、Emmett は最終的に実用的なプロトタイプを完成させました。彼は、依存関係の逆転を実現するために UEFI 標準の一部を使用し、テストダブルを手作業でコーディングし、粗い C 単体テストフレームワークを移植し、結果を Windows ワークステーションまたは CI サーバーに戻すためにシリアル接続で基本転送プロトコルを構築し、構築ツールチェーンまたは努力を指揮したのです。

私は、技術的にそれほど難しくない状況で、命令型言語で開発するエンジニアが、自動ユニットテストが技術的に不可能である理由として、あらゆる種類の理由を提供するのを聞いたことがあります。しかし、それを信じてはいけません。Emmett は、複雑な回路基板の原始的な泥の中で、あらかじめ構築された有用な単体テストツールのない状況で、単体テストの自動化を達成しました。

BIOS チーム数の増加

初期の採用範囲を管理しやすくするため、当初は最新のインテル製チップセットとリファレンス・ボード・デザインのサポートに携わる人たちだけを対象としていました。旧製品のインテル製チップセットに関するバグフィックスやその他のマイナーな作業に従事する者は、レガシー組織設計の中で作業を継続しました。つまり、BIOS コンポーネントの境界には、最新のプロトタイプ Intel チップセットだけが含まれていたのである。

Intel のメジャーリリースごとに AMI のコードベースが分かれていたため、この戦略はかなりうまくいきました。レガシーIntel チップセットに関する作業が一段落する頃には、最初の 3 つの BIOS スクラムチームは、ほぼ統合されていた。

サンフランシスコの BIOS 開発者が増えたため、BIOS コンポーネントバックログを 1 つにまとめたものをもとに、新たに 2 つのチームを結成しました。それぞれの新しいチームは、プロセスへの完全な関与とコミットメントを確保するために、様々なチーム横断的な成果物に影響を与え、承認することが許可されました。思い出せませんが、この取り組みの一環として、必要な人にはアジャイル概要トレーニングを実施したはずです。

2 つの新しい BIOS スクラムチームを立ち上げると同時に、BIOS コンポーネントの境界を事実上拡大し、Intel チップセットの全世代を含むようにしました。

BIOS コンポーネントの境界を拡大する

MCS システムの最上位には、MCS インフラストラクチャ全体を管理する責任を負う Modular Compute System Administrator (MCSA) があります。MCS の顧客のシステム管理者が MCS と対話するとき、それは通常、彼らが対話する MCSA です。

MCS の BIOS コードの強化のいくつかは、MCSA で小さな変更を行う必要がありました。また、各ノードで実行されている低消費電力ノードコントローラ上で実行されているファームウェア内で行われる必要がありますあまり一般的ではありませんでした。

当初、BIOS チームは、より上級の MCSA 開発者数名と、横断的な機能に関して共同作業を行いました。BIOS スクラムチームは、MCSA 開発者と作業を調整することがしばしば困難であることに気づきました。MCSA の開発者は、ウォーターフォール型のプロジェクトにつきものの、納期に対するプレッシャーに悩まされていた。

もし、政治的に可能であれば、MCS 製品全体の自然な要件領域と一致する BIOS コンポーネントの境界をより広く拡張して開始することで、上記のような調整の問題は存在しなかったでしょう。

政治的な状況を考慮すると、MCS BIOS 開発者が MCSA の変更を自ら行うための専門知識をゆっくりと身につけることが明白な解決策でした。BIOS スクラムチームが成熟するにつれ、BIOS スクラムチームの多くにとってこの必要性はますます明白になっていきました。私も、この方向で彼らを励ます傾向がありました。

様々な会話を通じて、私たちは、MCSA で働く少数のエンジニアと、ノード・コントローラ・ファームウェアで働く別のエンジニアから、賛同を得ることができました。この合意は、BIOS スクラムチームが、横断的な変更を BIOS スクラム自身が行えるほど簡単に行えるようにするための指針として、彼らに協力してもらうというものでした。その後、BIOS スクラムチームが作業を行い、BIOS チーム外の関連する専門家にコードレビューを行わせることになりました。

サンフランシスコの BIOS 開発者は、何年も前からサンフランシスコの MCSA 開発者の多くとうまくコラボレーションしていた。MCSA チームの主要なリーダーやアーキテクトの観点からは、BIOS チームが MCSA と BIOS の統合作業の一部を引き受けることは、歓迎すべきことだと考えられていたのです。ですから、ソフトウェア担当副社長が現状維持のために消極的に積極的に動いていたことは、実際には重要な問題ではないことがわかりました。MCSA と BIOS の統合作業が完了する限り、上級管理職の誰もそれがどのように行われたかを気にすることはなかったのです。

Node Controller の開発者は、診断チームの設立に携わったのと同じディレクターを通じて報告する。そのため、BIOS チームにクロストレーニングを施し、ノードコントロ

ーラ内で小さな変更を加えるというような、政治的な大きな課題はなかったのです。Node Controller の開発者のほとんどは、ポートランドの BIOS チームが使っていたのと同じポートランドの小さなオフィスに座っていた。

ミーチャと私は、このアプローチの成功は時間とともに大きくなると推論しました。いずれ、MCSA のエキスパートを 1 人か 2 人、BIOS グループに再配置しないことが、政治的に難しくなるだろうと予測したのです。BIOS スクラムチームが MCS を横断してエンドツーエンドで作業できるようになればなるほど、BIOS コンポーネントの境界はより自然なものになっていくでしょう。より多くのプロダクトバックログアイテムが、MCS プロダクトマネジメントグループにとって意味のあるものになっていくだろう。そうなれば、ミーチャは偽のプロダクトオーナーから脱却することができる。そして、チームのダイナミクスをさらに向上させ、私の代わりに効果的なスクラムマスターを育てることができるようになる。

しかし、少なくとも 1 つの BIOS チームを BIOS コードベースの外側に引き伸ばすという合意に対して重要な行動を起こす前に、私たちの周りで分裂が侵食され始めたのです。_【サポート体制の崩壊】(#bookmark=id.3118frh)_の項では、上級管理職の交代とレイオフについてもう少し詳しく説明します。

BIOS インドチームの挑戦

サンフランシスコとインドのオフィス間の時差が非常に大きいことも、常に課題でした。同様に、報告系統の違いもアライメントの問題を複雑化させる傾向にありました。以下に、そのいくつかを紹介します。

以下の LeSS フレームワークと LeSS Huge フレームワークのルールを振り返ることは有益である。

- 製品グループについては、「最初に」完全な LeSS 構造を確立してください。
- 構造的な変更を含む LeSS Huge の採用は、進化的な漸進的アプローチで行われます。

シニアレベルの経営陣の賛同が得られれば、インドチームとの協働で直面した問題のいくつかは、もっと早い段階で構造的に解決できたかもしれません。しかし、一方で、一度に多くの変化を起こすことは、持続可能なことでもあります。BIOS の LeSS 採用が開始されるまでに、積極的に参加した当初のエンジニアリング SVP が退職していたことを忘れてはならない。

BIOS のアジャイル導入を通して、ミーチャと私は、私たちが行っている仕事を広く社会化するよう心がけました。古い Intel チップセットのレガシーサポートに取り組んでいたエンジニアでさえ、新しい Intel チップセットで使用されている Definition of Done を認識していました。インドからのリードエンジニアは、LeSS 指向のスプリントレビューに参加するために、しばしば自宅からダイアルインしていました。

図 15 とその近くの *BIOS Organizational Context* セクションで、ミーチャと私がとった、インドにいる BIOS 開発者の組織変更を孤立させて遅らせる戦術を説明しました。拡大したマルチコンポーネントの境界の 1 つとして Intel チップセットの世代を使うことができたことと、インドが当初は旧世代の Intel チップセットに関する問題の解決に注力していたことが、これを可能にしたのです。この戦略を完全に実行する前に、ハードウェア担当の副社長が退職してしまったため、サポートの継続性に依存するという点でリスクの高い戦略でしたが、私たちの制約を考えると、他に実行可能な選択肢は思い当たりません。もし他に選択肢がなければ、将来この戦略を繰り返すこともあるかもしれませんが、可能であれば、より広範な変化への賛同を前もって得ておくことを強くお勧めします。

コーチングサポート

MCS 部門では、最初の 1 年間は私一人のコーチでした。トレントと私はかなり早い時期にインドを訪れましたが、当初は現地に大きな影響を与えるほどのコーチング能力を持ち合わせてはいませんでした。私は数日間のトレーニングを行い、人間関係を構築しましたが、それ以外はほとんど実践的ではありませんでした。

2 年目に、Trent はインドで数人のコーチを雇うのに十分な資金を得ることができました。インドでベテランのアジャイルコーチを見つけるのは、米国で見つけるよりもさらに難しい。しかし、何ヶ月も面接を重ねた結果、私は最終的に数人を見つけることができた。さらに 1 カ月ほどして、Trent と私は、これらのコーチのためにインドのマネージャーの共同承認を得ることに成功し、交渉とオンボーディングを終えました。

インド拠点の BIOS エンジニアに対するウォーターフォールの圧力

レガシーIntel チップセットに関する作業が先細りになるにつれ、インドにいる数人の BIOS エンジニアが新しい Intel チップセットの BIOS コードベースへの貢献を開始しました。インドのエンジニアは、LeSS 指向の BIOS チームの共通の「完了定義」や関連する規範を十分に理解していた。しかし、残念ながら、彼らのマネジメントは、まだウォーターフォール方式で進められていた。

サンフランシスコの BIOS スクラムチームの初期メンバーは、自分たちが苦労して向上させた品質を落とさせまいと、断固とした姿勢で臨んだ。サンフランシスコの BIOS チームのメンバーは、継続的インテグレーションを改善し、統合テストの自動化を進め、あらゆるコード変更を頻繁にレビューし、インドの BIOS エンジニアと何度も腹を割って話をした。これらの戦術的な解決策は、間違いなくプラスの効果をもたらし、問題のいくつかを部分的に軽減することができました。

ミーチャと私は、これらの問題を予期していました。私たちは、政治的な勢いが増すまで、組織的な戦いを遅らせるために、BIOS コンポーネントの境界を意図的に設計していました。この意図的な遅延により、インドオフィスでは有能なアジャイルコーチが確保され、米国に拠点を置く BIOS LeSS チームは結束し、BIOS LeSS チーム内で品質と予測の透明性が大幅に改善されたことを示す重要な証拠があったのです。

ミーチャ、Trent、そして私は、根本的な構造的問題を解決するためには、かなりの努力とシャトル外交が必要であることを認識していました。私たちは、構造的な問題を解決するために、危機的な状況に追い込む覚悟はできていました。しかし、残念なことに、ハードウェアを支える副社長をほぼ同時に失い、大規模なレイオフが行われたことで、私たちの戦略は失敗に終わったのです。

アジャイル導入タイムライン」図、「組織構造」図、「MCS ハードウェア生成による BIOS 拡張コンポーネントチーム拡張」図を組み合わせて使用すると、これらのさまざまな要素の合流を見ることができます。

サポートシステムの崩壊

私がナカシマ MCS 部門に在籍していた頃、報告体制、部門資金、従業員数、経営陣が大きく変わりました。

市場シェアが低下し、競争環境が成熟化する中で、MCS の売上は縮小していました。ナカシマの経営陣は、MCS 部門を抜本的に縮小することを決定した。その結果、事業部の人数は約半分になった。ナカシマの多くの人が、「ナカシマで見たこともないような大規模なレイオフだ」と言っていた。少なくとも BIOS チームの半数は、すぐにレイオフされることになる。

レイオフに先立ち、管理職層の多くが、資金力のある新興企業に引き抜かれることになった。その中には、ミティアがもともと所属していた副社長も含まれていた。ミーチャはその後、もともとスクラムとアジャイルを完全にサポートしていなかったソフトウェア担当副社長に報告をするようになった。私が退職して間もなく、ミーチャは前の副社長に続いて新しいスタートアップ企業に行った。

私はレイオフ後も残りましたが、Trent が私の契約を更新することは政治的に不可能になりました。アジャイルコンサルタントの相場と旅費は、少なくとも 2、3 人のエンジニアを雇うのに十分な金額です。Trent に任せていれば、私は契約を延長してもらえたかもしれませんが、Trent でさえ、2 つ上の層の予算の承認を得なければならなかったのです。それから 1 年余り後、トレントもナカシマを去った。

結論

組織的変化に関する考察

振り返ってみると、エンジニアリング担当の SVP が早期にいなくなったことと、短期的な利益率の急回復に焦点を当てた頻繁なレイオフの文化で長期的な雇用の安全が確保できなかったことが、組織構造と文化の変革で長期的に成功する確率を劇的に低下させたのです。これらの課題がなければ、BIOS コンポーネントでの成功は、より広範な LeSS 導入に活用できたと私は考えています。

BIOS の LeSS 導入が成功した後、当初のエンジニアリング SVP は、さらに上を目指すために必要な上級管理職の幅広い参加に貢献できたはずだと私は考えています。その代わりに、シニアマネジメントは、差し迫ったレイオフという政治的な状況を切り抜けようと奮闘しながら、近々リリースされる MCS を押し出すために奔走することになりました。もっと早くから積極的な構造改革を行っていれば、非常に有効だったと思いますが、私たちが時間をかけてゆっくりと実績を積み上げていかなければ、関係する上級管理職はそれを受け入れる準備ができていなかったと思います。

種をまく農夫が、さまざまな土壌に広く種をまくというたとえ話を思い出しました。浅い土にまいた種は発芽も成長も早いが、根が張らないのですぐに太陽の下で枯れてしまう。一方、深い土にまいた種は、豊かな実りをもたらす。LeSS を導入する際に土壌を適切に整えなければ、初期には劇的な成功を収めることができても、長期的には問題が発生する可能性が高いのです。BIOS の LeSS 採用のための土壌は、初期の素晴らしい結果を生むには十分な深さでしたが、その豊かな潜在能力を発揮するにはまだ十分ではありませんでした。

特典の概要

BIOS チームは、拡張コンポーネントチームから始まり、_拡張コンポーネントチーム、フィーチャーチームへと移行していきました。利点は以下の通りです。

- 拡張されたコンポーネントバウンダリー内で、ローカルに改善された**適応性**と**価値提供**を実現する。
  - 事前に観察したこと
    - ある BIOS のサブコンポーネントを理解しているのは、1 人か 2 人しかいなかった。コードの所有権と知識は、事実上、個々のファイルレベルであった。
    - BIOS コンポーネント以外のコードベースに関する知識は、BIOS エンジニアリンググループ内にほとんど存在しなかった。
    - 新しいブレードハードウェアで新しい Intel CPU の世代を立ち上げるために必要な作業の全体を把握している人はいませんでした。BIOS ファームウェアエンジニアと BIOS ファームウェアテスターは、それぞれ、パズルの小さな一片を理解していただけでした。
  - 以前の影響
    - BIOS エンジニアは、個々の専門領域に制限されていたため、どの時点でも、最も価値のあ る作業を選択することができなかった。
    - 完成を正確に予測することは、ほぼ不可能だった。

- その後の観察
  - すべてのチームメンバーは、以前より多くの BIOS サブコンポーネントを扱うことに徐々に慣れてきた。
  - 多くの BIOS チームメンバーは、GUI と BIOS コンポーネント間の実行パスに沿って、システム全体のあらゆる面をテストし、修正することにますます熟練するようになっていました。
  - BIOS チームのメンバーは皆、新しい Intel CPU を立ち上げるために必要な作業について、全体的によく理解していた。LeSS 指向の BIOS チームのどのメンバーも、BIOS コンポーネントのバックログにあるどの項目も、それが全体像にどう当てはまるかを説明することができました。
  - BIOS コンポーネントバックログは、次期 BIOS リリースの出荷に必要な既知の残作業のすべてについて、常に最新の情報を提供してくれました。
  - BIOS チームは、製品開発の自然な変動性を受け入れ、BIOS Definition of Done に明示されているように、（スコープではなく）クオリティにコミットした。
- その後の影響
  - LeSS 指向の BIOS チームは、どの時点でも一貫して最も価値のある仕事に集中することができた。
  - BIOS チームは、リリース予測にはるかに自信を持つことができた。
  - BIOS チームは、優先順位の低い機能に取り組む前に、リリース可能なレベルの機能が達成されたことを確認することができた。
  - LeSS 指向の BIOS 拡張コンポーネントチームは、すべての Sprint Increment が MCS システム全体とうまく統合され、テストされたことを確認した。数千人のエンジニアの中で、同様のことを主張できるのは、小規模な診断パイロットスクラムチームだけでした。

・ 品質向上**を実現した技術的実践の改善と、さらなる改善への意識の向上

- 事前に観察したこと
  - 品質基準がどのようなものであるかについての共通の合意がなかった。
  - AMI から提供された BIOS コードの変更は、MCS 固有のコードと混在しており、メンテナンスが困難だった。新世代の Intel CPU は、事実上ゼロから始める必要があった。
- 以前の影響
  - MCS を新世代の Intel CPU に適合させるために、1 年以上と BIOS エンジニアの軍団を要した。インテルが公式に新しい CPU をリリースする前にこの作業を完了できなかった場合、数千万ドルの収益損失が発生する。

- その後の考察
  - BIOS の Definition of Done は、各新 BIOS チームによって受け入れられている生きた品質基準である。Done の定義は、チーム自身によって時間をかけて継続的に明確化、拡張される。
  - BIOS Definition of Done には、MCS BIOS の変更が可能な限り AMI プラグインレイヤーを活用することを保証するコミットメントが常に含まれている。この賢明な要件は、監督や経営陣からではなく、チーム内部から生まれたものである。
- その後の影響
  - 技術的な慣習が改善されたことで、次世代 Intel CPU への対応に要する時間が劇的に短縮されることが期待されます。
  - 健全なレトロスペクティブと品質重視の姿勢は、次の改善策を常に明確にする。

- 拡張部品に関連する**不具合**の早期発見と解決.
  - 事前に観察したこと
    - BIOS ファームウェアのエンジニアは、コードの変更を行い、そのコードを、テストグループ に丸投げしていた。テスト担当者が問題を特定し、BIOS ファームウェアエンジニアに問題を投げ返すまでには、通常、数週間から数ヶ月を要した。
    - 自動化されたテストは、ほとんどありませんでした。ほとんどすべてのテストは、非常に手作業で行われていました。
    - コンパイルに問題があるコードをソースコントロールにコミットするのは、一般的な慣習でした。
  - 以前の影響
    - 不具合が発見され、解決されるまでには、通常数ヶ月を要しました。
    - BIOS エンジニアは、他の開発者によってもたらされた破壊的な変更を追いかけるのに、しばしば時間を浪費していました。多くの場合、この作業は、問題が発見されたときにはもう仕事をしていない、離れた時間帯にいる別の開発者によって行われました。
  - その後の観察
    - どの BIOS チームにも、開発および変更のテストを行うためのスキルと知識を持った人がいました。
    - 各 BIOS チームは、通常、一度に少数の変更を行うことに集中していた。彼らは、個人としてではなく、実際のチームとして、関連する製品バックログ項目の Definition of Done を達成することに積極的に焦点をあてて働いていた。
    - BIOS チームのメンバーは、技術的な知識と専門的な知識の両方において、積極的にお互いを鍛え合いました。テスターはより強い開発者になり、開発者はより強いテスターになりました。

        - 自動テストのレベルを上げ、自動ビルドシステムを導入することで、問題の早期発見をより確実なものにしました。
        - 自動化されたビルドとテストは、従来のチームと LeSS 指向のチームとの間で矛盾する優先順位を特定するのに特に役に立ちました。
    - その後の影響
        - 欠陥の発見と解決は、数ヶ月ではなく、数分から数時間以内に行われました。コードフリーズ」後のバグ潰しで何ヶ月も無駄にするのが常だったが、それが全く不要になった。
        - レガシー構造の中で働いていた海外チームの品質に対するインセンティブや目標の矛盾は、海外チームが同じコードベースで仕事をするようになると、非常に見えやすくなりました。時間とマネジメントの継続性があれば、この可視化によって、海外チームの再編成と改善も政治的に可能になっただろうと思う。
- 拡張コンポーネントチーム内の従業員のコラボレーション、エンゲージメント、学習**の改善
    - 事前に観察したこと
        - チームというより個人として働き、各自が個人的な課題に集中していた。
        - チームは、マネージャーが交流や仕事の割り当てを主導・推進する典型的なエンゲージメント・モデルに従っていた。
        - 成長と学習は、個人の責任範囲と過去の専門知識の範囲に限定されていた。 自分の専門性と責任を新しい分野に広げるには、仕事をアレンジし、シフトさせる必要がありました。これには時間がかかり、必ずしも現実的ではありませんでした。
    - 以前の影響
        - 「多くの努力と限られた見返り」-ミーチャ
    - その後の観察
        - チームは自分たちで仕事を管理し、自分たちのチームが選んだ PBI を実現することに集団的かつ熱心に集中していた。
        - 個人もチームも、自分たちのチームが今集中している PBI を完成させるために必要なことは、何でも積極的に学んでいた。チーム内、チーム間の知識の伝達の度合いは、以前観察されたものより桁違いに大きい。
    - その後の影響
        - Richard Hackman 氏の「Leading Teams」の用語では、BIOS チームは、緩やかに協力する個人ではなく、「真のチーム」として行動していた。各チームは、明確なチームタスク、明確な境界線、自身の作業プロセスを管理するための明確な権限、そして長期にわたるメンバーシップの安定性を持っていました。

- 組織的な阻害要因**に対する認識が深まり、さらに組織的な変革が必要になっている
    - 事前の観察
        - 開発サイクルが長い、不具合発見が遅い、チーム間で頻繁に引き継ぎがある、一人の作業がボトルネックになるなど、様々な問題が普通に受け止められていた。もっと良くなるはずだという意識が希薄。
        - 個々のチームメンバーには、グローバルレベルでの適応性や価値提供に注力する構造的な動機がほとんどなかった。
        - 長年にわたる組織の障害を取り除くための議論や努力はほとんどなされていなかった。
    - 以前の影響
        - グローバルレベルでの適応性と価値提供を改善するための生産的な努力はほとんどなかった。
    - その後の観察
        - BIOS チーム内の定期的なチームおよび全体の振り返りにより、組織的な障害が頻繁に特定され、チームと経営陣はそれを積極的に解決しようとするようになった。
        - BIOS の変更を完全に検証するための BIOS 拡張コンポーネントチームの努力は、コード所有権の境界を挑戦し、複数のコンポーネントにまたがってコード変更を行う能力を拡大させるようになった。
        - BIOS Definition of Done で明示されたより厳格な品質基準の追求は、レガシーBIOS チームの構造的なインセンティブの矛盾を浮き彫りにするのに役立った。
    - その後の影響
        - 人々は、何が可能か、そしてより大きな組織変革の必要性に徐々に目覚めていった。

ラップアップ

私は、長期的な成功を収めるには、製品の境界を広く取り、製品の全範囲で実行するフィーチャーチームが最適だと考えています。もし「可能性の芸術」がこの選択を排除するのであれば、拡張コンポーネントチームから始めて、拡張コンポーネントチームと機能チームに積極的に移行する漸進的なアプローチは、まだ追求する価値があります。

謝辞

編集審査の全過程を通じて、Viktor Grgic 氏および Craig Larman 氏より多大なご尽力とご指導を賜りましたことに、深く感謝申し上げます。

オンライン版の翻訳原稿をご提供くださった清水皓貴氏に、心より感謝申し上げます。また、書籍化に向けた翻訳作業の過程で、私の原稿の校閲にご尽力いただいた清水皓貴氏および丸田恭平氏にも、深く感謝いたします。

日本語版 LeSS コース

日本の LeSS コミュニティをより一層支援するため、対面形式で実施している私の「Certified LeSS Practitioner（認定 LeSS プラクティショナー）」および「Certified LeSS for Executives（認定 LeSS エグゼクティブ）」コースについて、全面的に日本語化されたバージョンをご用意いたしました。これらのコースは、公開講座としても、また企業・組織向けのプライベート講座としてもご受講いただけます。

私自身は日本語を話すことができませんが、講座の実施にあたっては、多彩な通訳者がライブ通訳（逐次・同時通訳）を担当いたします。通訳陣には、国際会議通訳者協会（AIIC：https://aiic.org/）に所属するプロフェッショナルに加え、LeSS コミュニティ内のバイリンガルメンバーも含まれております。

詳細につきましては、過去に東京近郊で開催された公開講座「Certified LeSS Practitioner」の開催概要ページを以下のリンクよりご参照ください。

https://agilecarpentry.com/clp/jp_tokyo_summer_2025/

連絡先

お忙しい中、本ケーススタディをお読みいただき誠にありがとうございます。ご質問やご感想などございましたら、どうぞお気軽にご連絡ください。貴組織の改善に向けた取り組みに、微力ながらもご協力させていただける機会を心より歓迎いたします。

James Carpenter

Certified LeSS Trainer

Cell: +1 832-677-724

メール: *james@agilecarpentry.com*

Web サイト: *https://agilecarpentry.com*

LinkedIn: *https://www.linkedin.com/in/jamescarpenter1/*