GUI 開発におけるアーキテクチャドリブンな インクリメンタル開発の提案と自動車 HUD ソフトウェア開発への 適用評価

白木 徹[†1]　林 健吾[†1]　青山 幹雄[†2]

概要：GUI アプリケーションなどのソフトウェアシステムは，MVC (Model View Controller)アーキテクチャパターンを採用している．著者らが従事する自動車の HUD ソフトウェア開発では，システム試験開始時にシステム全体を統合するビッグバン統合を行っていたため，システム試験に高い負荷がかかり，開発量の増加に伴い納期と品質の確保が困難となっていた．この問題を解決するために，MVC アーキテクチャに基づきインクリメンタルに開発するプロセスを提案する．提案開発プロセスでは，MVC の中で View の開発量が多いことと，View のコンポーネントの独立性が高い点に着目し，継続的統合を導入して開発負荷を平準化する方法を開発した．提案開発プロセスを自動車の HUD ソフトウェア開発へ適用し，納期遵守と品質向上を実現したので報告する．

An Architecture-Driven Incremental Development Method for GUI Applications and its Evaluation in Automotive HUD Software Development

TORU SHIRAKI[†1]　KENGO HAYASHI[†1]　MIKIO AOYAMA[†2]

1. はじめに

自動車システムの複雑化に伴い運転者に提供する情報量が増大している．運転者に情報を提供する手段としてフロントガラスに情報を投影するヘッドアップディスプレイ(HUD: Head Up Display）を用いたユーザインタフェースの導入が，高級車から中級車へと拡大し，さらに低価格帯の車両へと拡大している[6]．HUD は，搭載車両の拡大と共に，表示するコンテンツ量も急増し，そのソフトウェア開発量も急増している．

著者らは，このような HUD のソフトウェア開発に従事している．HUD ソフトウェアは GUI 開発が中心となり，ユーザへの表示コンテンツが多様であることから，MVC (Model View Controller)アーキテクチャパターン[4][9]を採用している．MVC の 3 つのサブシステムの開発は必要となるスキルやツールセットが異なり，開発組織もサブシステム毎に異なる部門が開発を分担している．このように，分担開発を行っていることから，システム試験のためのソフトウェア統合方法としてビッグバン統合（以下，BI と略記）を採用していた．しかし，開発量の拡大に伴い，BI ではシステム試験における工数が増大し，品質確保が困難になり，納期が遅延するリスクも生じていた．特に，商品性確保のために View の仕様変更の頻度が高く，ソフトウェア変更後の妥当性確認はソフトウェアの結合後にしか行えない．BI では開発量の増大と共に統合時期が開発プロセスの最終段階に遅延する傾向があり，品質保証の工数と期間が確保できなくなる．

従って，品質保証の工数と期間を確保するために BI から継続的統合（以下，CI と略記）[2]によるインクリメンタル開発方法に移行したい．しかし，MVC の各サブシステムを異なる組織で異なる開発速度，開発粒度で開発する場合のインクリメンタル開発方法は未確立である．

本稿では MVC アーキテクチャドリブンによる GUI のインクリメンタル開発方法を提案する．MVC アーキテクチャでは View の開発はコンテンツごとに独立に実行できるが，他のサブシステムに比べ開発量が大きいことから View を中心とする開発プロセスと統合方法を採ることとした．提案方法を自動車 HUD ソフトウェアシステムの実開発に適用し，バグ検出の早期化と工数の平準化，および，バグ検出の早期収束を実現し，納期遵守と品質向上を実現した．

2. 研究課題

従来の BI による開発プロセスは，品質低下と開発効率悪化の 2 つのリスクがある．

品質低下リスクは MVC アーキテクチャによる GUI アプリケーションの BI において生じている．BI による開発プロセスでは，結合テストは Model，View，Controller の各サブシステムの開発が完了するまで実行されない．その結果，バグの検出が開発工程の終盤に持ち越され，ソフトウェアの修正期間が十分に確保できず，納期遅延や品質低下が生

†1 (株)デンソー
DENSO CORPORATION
†2 南山大学
Nanzan University

じ得る．

開発効率悪化のリスクは開発組織構造と BI の結合頻度に起因する．結合テストを実施するまでは，設計者と実装者の作業負荷が高い．一方，結合テスト以降はテスト/評価の担当者の作業負荷が高くなる．設計者，実装者，テスト/評価者を異なる組織が分担する場合は，作業負荷のバラつきに対して組織間でリソースを融通し合うことができないため，開発効率の低下をもたらす．

これらの問題を解決する方法としてインクリメンタル開発プロセスの導入が考えられる．インクリメンタル開発プロセスにより設計，製造，検査を段階的に行い，開発の作業負荷の平準化をすることが可能となる．品質低下リスクに対してはバグを早期に発見し，修正期間を十分に確保することが可能となるため，納期の遵守と品質確保が可能となる．開発効率低下リスクに対しては，開発の作業負荷を平準化し，開発効率を高めることができる．

このため，開発対象が MVC アーキテクチャであることに着目し，MVC ぞれぞれのサブシステムで分割された開発組織構造に適合するよう開発プロセスを変更してインクリメンタルな開発に移行することで，上述の問題を解消することが本研究の課題である．

3. 関連研究

3.1 リスクドリブン開発プロセス

リスクドリブン開発プロセスは，ソフトウェア開発で生じるリスクを予測し，リスクを最小とする開発方法を採用する開発プロセスである[1].

スパイラルモデルは，リスク分析とプロトタイプ開発を繰り返し行うことでリスクを最小化し，品質を高める開発プロセスモデルである[1]．検証可能なプロトタイプを構築することで，要求仕様や設計などの開発工程で混入する様々なバグを早期に発見することが可能となる．

3.2 継続的統合(CI: Continuous Integration)

BI は，ソフトウェア変更後の妥当性確認はソフトウェアの結合後にしか行えない．そのため，バグの検出は開発工程の終盤となり，ソフトウェアの品質を高めにくい． CI は，BI の欠点を解消するため，実行可能ソフトウェアとして継続的に統合することで，バグを早期に検出し，修正することでソフトウェアの品質を高めるためのアプローチである[2]．開発の序盤に統合開始することで，ソフトウェア変更後の妥当性確認に必要な時間の確保が可能となる．

3.3 アジャイル開発

アジャイル開発は，開発を機能単位で細分化し，分析，設計，製造，検査のサイクルを機能毎に実行し，インクリメンタルに開発する方法である[5][8]. 開発は顧客にとって価値の高い機能から開発し，ソフトウェアは常に動く状態を維持し続ける．さらに，仕様変更に柔軟に対応できる特徴がある．機能を独立して開発できるアーキテクチャを採用している場合に有効である．

大規模なアジャイル開発のためのフレームワークとして SAFe がある[3]．SAFe では，機能を追加しやすくするために，機能開発とは非同期にアーキテクチャを拡張する Architecture-Runway を提案している．プラットフォーム開発のように複数の機能が依存するソフトウェア部品を開発する場合に有効である．

4. アプローチ: アーキテクチャドリブン開発プロセスの考え方

ソフトウェアの開発活動を決定づける要因の関係性を明らかにするフレームワークとして，図 1 に示す BAPO（Business，Architecture，Process，Organization）フレームワークが提唱されている[7]．BAPO では，これら 4 つの要素が互いに作用し合い，また互いの要素に適合していることが望ましいと考える．BAPO に照らし合わせた場合，現在の MVC アーキテクチャは多様性を備えた GUI 製品を提供するという Business に適合させた結果であると言える．一方，開発組織は MVC アーキテクチャに合わせて 3 層に分割される．開発プロセスはアーキテクチャと組織には適合しておらず, 従来のウォータフォール開発による BI を継承していた．本稿では，開発プロセスをアーキテクチャに適合させるために，アーキテクチャドリブンな開発プロセスに着目するアプローチを採る．

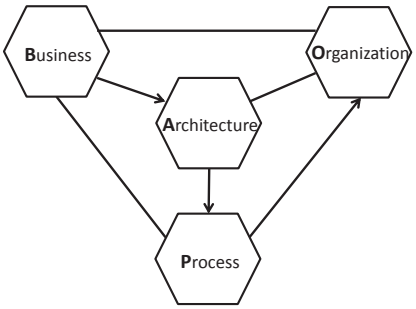

図 1 BAPO の概念図[7]

5. アーキテクチャドリブンなインクリメンタル開発プロセス

5.1 HUD の MVC アーキテクチャと開発プロセス設計方針

5.1.1 HUD の MVC アーキテクチャ

開発対象の HUD の MVC アーキテクチャの特徴を図 2 に示す．Model では車両通信の内，表示要求のあるコンテンツを Controller に渡す機能を提供しており，車両通信の変化を関心事としている. View では製品ごとに表示方法を切り替えるための表示を提供しており，製品グレードの見映えの変化を関心事としている. Controller では表示するコンテンツ情報間の相互作用をコンテンツの優先度に従って調停する機能を提供している．

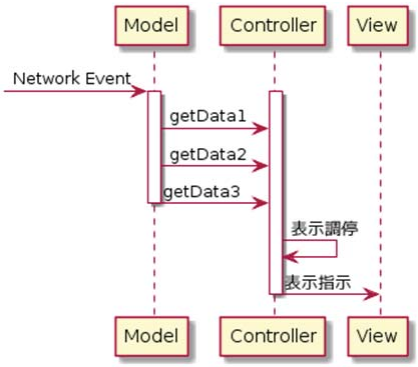

図 2 HUD の MVC アーキテクチャ

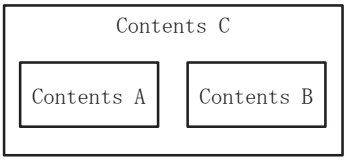

図 3 表示の優先順位例

例えば, 図 3 のように Contents C の表示領域に Contents A, B の表示領域を内包する表示エリアを想定する．表示の優先順位は{A, B}＜C とする．この時，Model から Contents A と Contents C の表示要求があった場合，Controller は調停の結果，高優先である Contents C を選択し，View に渡してコンテンツを表示する．

また，サブシステムの開発には 2 つの特徴がある．一つ目の特徴は, Model や View はコンテンツごとに独立して開発することができ, Controller は調停すべき全てのコンテンツを対象に一括して開発する必要があるということである．Controller の開発は分割が困難である．

二つ目の特徴は，運転者に情報を提示する特性上，製品個別で View を変更したいという顧客要求が強く，View の開発量が増加傾向にあることである．また，View の仕様変更の頻度は高く，画面のちらつきやアニメーションなど，ハードウェアに関連するソフトウェア変更の妥当性確認はソフトウェアの結合後にしか確認できない．

5.1.2 アーキテクチャドリブン開発プロセスの設計方針

HUD のアーキテクチャの特徴を考慮して開発プロセスの設計方針を 2 つ策定した．

(1) 開発の方針

Controller の開発を優先的に完了する方針とする．MVC アーキテクチャでは，View 間に依存関係が少なく，View の画面ごとに独立した開発ができるため, View ではインクリメンタルな開発が可能である．同様に，コンテンツ間の依存関係が少なくコンテンツごとに独立した開発ができるため，Model ではインクリメンタルな開発が可能である．一方，Controller では Model と View の中間層として Model と View の動作変更を理解し吸収する複雑なソフトウェア開発が必要となるため，インクリメンタルな開発はできない．しかし，サブシステムの開発の特徴を活かして，MVC 開発の初期に Controller を開発することで HUD をインクリメンタルに開発することが可能となる．

(2) 統合の方針

ソフトウェアの統合は非同期として, View の開発速度を優先する方針とする．開発組織も開発速度も異なるため，MVC のすべてのコンポーネントの統合タイミングの同期は合理的でない．View の開発量が最も大きくなる特徴を考慮し, View の開発速度が最大化するようにソフトウェアの統合方法を設計する．

このような設計方針に基づいて，アーキテクチャドリブンなインクリメンタル開発プロセスを設計する．

5.2 アーキテクチャドリブンの開発プロセス

5.1 節で述べた特徴と設計方針から，Architecture-Runway の考え方と Queuing 統合を導入してアーキテクチャドリブンな開発プロセスを提案する．

5.2.1 Architecture-Runway の応用

図 4 に示すように，Architecture-Runway の考え方を応用し，インクリメンタル開発のベースとなる Controller を序盤に開発する．ひとつのコンテンツを実現する View と Model の開発の組み合わせを Application Group（以下，AG と略記）とする．Controller の開発後は Model と View それぞれで AG を同期した順序で開発することでインクリメンタル開発が可能となる．

5.2.2 QI(Queuing 統合)の導入

Model と View はインクリメンタルに開発する．しかし，View における個々のコンテンツに対する開発量は Model よりも大きく，統合周期に開発のタイミングを合わせようとすると Model の開発に待ちが発生して効率的でない．

そこで，統合を同期せず View の開発を先行して進めることのできる統合方法として Queuing 統合（以下，QI と略記）を提案する．QI では，統合して機能を実現するサブシステムは，開発順序だけ同じくし，開発完了のタイミングは同期させなくてもよい．開発完了したサブシステムは，対象コンテンツごとに統合対象のキューに加えられる．特

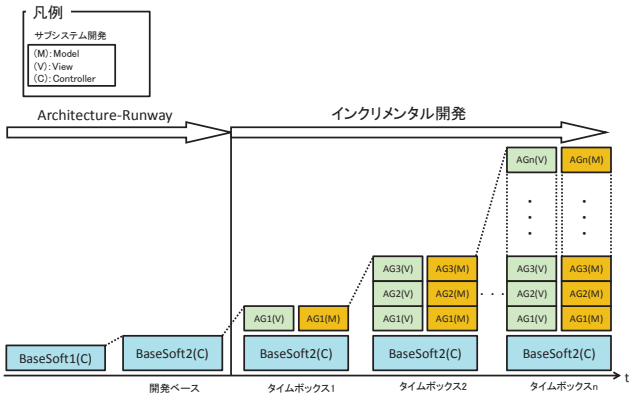

図 4 Architecture-Runway の例

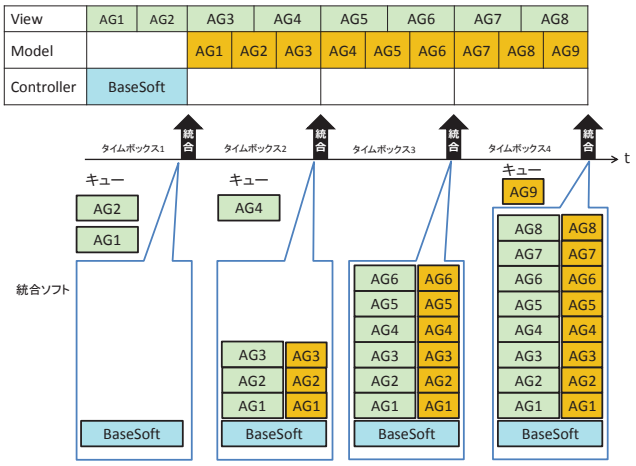

図 5 Queuing 統合の例

定のコンテンツを実現するすべてのサブシステムがそれぞれのキューの先頭に揃ったところで，システムとして統合し結合テストを開始する．統合する対象をサブシステムに限定することで，結合テストで発見されるバグを解析する対象も限定でき，統合後の工数も最小化することが可能となる．

QI による統合例を図 5 に示す．Model と View の開発の順序は AG1 から AG9 まで同じとするが，開発速度は異なる．統合のタイミングはあるタイムボックスの開発を行う中で最も開発の遅いサブシステムの開発が終わったときである．例えば，タイムボックス 2 の開発では，View は AG3 と AG4 の開発を行える．一方，Model は AG1，AG2，AG3 の開発を行える．そのため，開発の最も遅い Model に合わせて統合を行うため，AG1，AG2，AG3 が統合され，View の AG4 は統合されずキューに加えられる．タイムボックス 3，タイムボックス 4 でも同様に Model と View は独立して開発していく．開発の過程で View の開発が Model の開発と逆転して，Model の開発が先行した場合は，開発の遅い View の開発が終了した時点で統合することとなる．

QI を導入することで，異なる組織で開発したサブシステムのインクリメンタルな開発が可能となり，開発に時間のかかるサブシステムを先行して開発を行うことでシステム全体としての開発速度を上げる．

6. HUD 開発への適用

提案した開発方法を，HUD の実開発に適用する．QI の導入を容易にするために，アジャイル開発と同様に固定期間を開発リズムとするタイムボックス[5][8]を導入した．タイムボックスを導入することで，統合は周期的に固定曜日で行え，リソースの割り当ても含めて管理コストを抑えることができると考えた．著者らの開発グループではインクリメンタル開発を初めて導入するため，開発に慣れることを目的としてタイムボックス期間を 1 週間に設定した．従来の開発プロセスの BI で同規模の開発をした場合，開発期間は 3 ヶ月程度となる．そのため，12 タイムボックスの計画を立案することとした．

タイムボックスごとに優先順位に従って，開発対象を選択して開発した．開発の優先順位は下記ルールに従う．優先度が高い順にルールは並べている．

(1) BaseSoft となる Controller を優先して開発する．
(2) インクリメンタル開発に慣れるために開発が容易な機能から開発する．
(3) 移植など変更規模が少ない AG から開発する．
(4) 変更開発，新規開発となる AG を開発する．

これらの開発の優先順位に従って作成した開発計画を図 6 に示す．図に示すようにベースとなるソフトウェアの開発を先行して行い，それ以降は各 AG をインクリメンタルに開発していく．このようにすることで開発が終わった AG は次のタイムボックスで評価を実施することができる．

例えば，タイムボックス 2 の統合時には AG1，AG2，AG3 の開発が完了しているため，タイムボックス 3 で AG1，AG2，AG3 の評価をすることが可能となる．このようにすることで，検査を早期化することができ，ここで検出したバグは次のタイムボックスで修正することができる．

さらに，インクリメンタル開発をタイムボックス毎に行うことで，システム試験の評価をタイムボックスに分散することができる．その結果，システム試験の評価期間を短縮することができる．

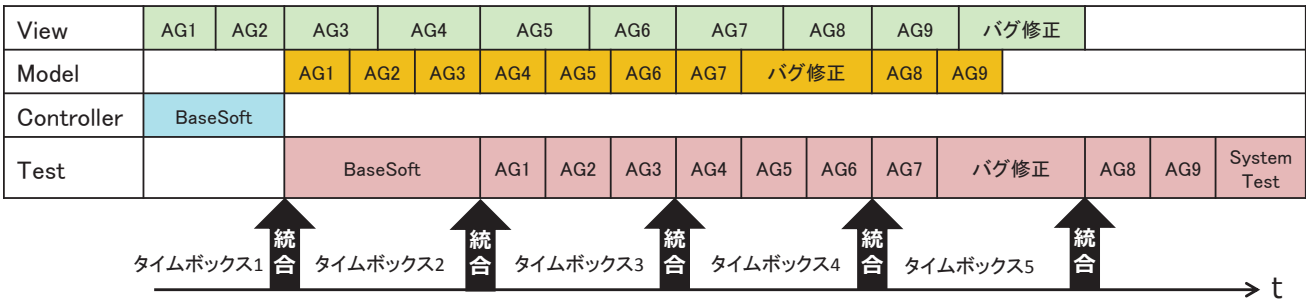

図 6 開発計画の例

7. 評価

7.1 評価環境

提案方法の適用評価として，品質と開発効率の向上の 2 つの効果を評価する．

品質向上効果については，バグ検出の早期化とタイムボックス毎のバグ検出数の平準化を評価するためにバグ曲線を計測する．また，検出したバグの内容を分析するため，バグ検出数を分類する．

開発効率向上の効果については，開発工数が設計，製造，検査の工程が開発チームで平準化されていることを評価するために，各月毎の工程別の開発工数の推移と月毎の開発総数，サブシステム毎の開発工数を計測する．さらに，変動係数を算出することで開発工数のばらつきを比較する．変動係数とは，標準偏差を平均値で割った値のことである．データのばらつきを相対的に評価する指標であり，値が 0 に近いほどばらつきが小さい．

比較対象は同一の開発車両，同一の開発メンバーにおける，提案方法の適用前とする．但し，開発規模は異なっている．

7.2 評価結果

7.2.1 バグ検出の早期化

図 7 に提案方法の適用前におけるバグ曲線とバグの内訳を示す．適用前では，バグの検出時期が納入の 2 タイムボックス前であることが分かる．また，バグの内容は，単体テストが多くを占めており，ソフトウェアの修正期間が十分に確保できていないことが分かる．

図 8 に提案方法の適用後におけるバグ曲線とバグの内訳を示す．適用前に比べて，バグ検出が早期化できていることが分かる．タイムボックス 2 でリグレッションテスト 5 件，タイムボックス 5 以降では各タイムボックスで単体テストのバグを検出し，計 11 件検出できていることが分かる．バグ総数 23 件中 16 件はソフトウェアにおけるバグであり，ソフトウェアの修正が必要となるが，バグ検出が早期化されたことで修正期間を確保できている．また，バグの検出方法は，適用前よりも多く，各アクティビティのバグ検出能力が十分に発揮できており，品質が向上できている．

以上のことから提案方法を用いることでソフトウェアを早期に修正することができ，納期の達成と品質の向上を実現できた．

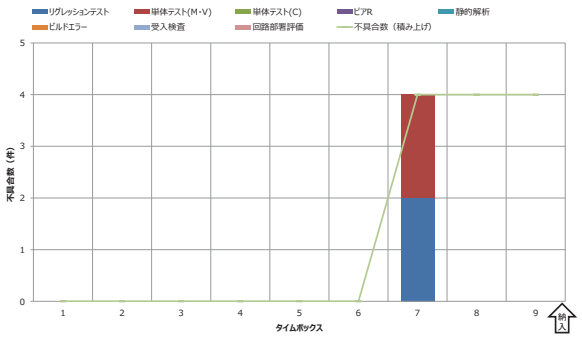

図 7 バグ曲線とバグの内訳（適用前）

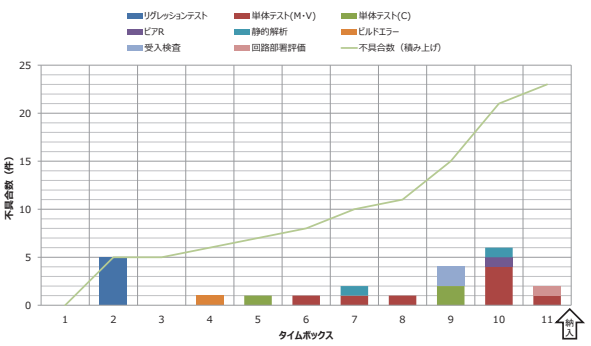

図 8 バグ曲線とバグの内訳（適用後）

7.2.2 開発工数の平準化

図 9 に適用前の月毎の工程別の開発工数の推移を示す．適用前では，8 月と 9 月に要求分析から設計，製造の大部分を行い，9 月と 10 月で検査を実施していることが分かる．また，全体の開発工数では，検査を多く実施している 9 月が最も多く，開発の後半に検査工数が集中している．

図 10 に適用後の月毎の工程別の開発工数の推移を示す．適用後では，月毎の要求分析，設計，製造，検査の工数が平準化され，全体の開発工数においても平準化されていることが分かる．

図 11 に月毎の開発総数と変動係数の比較結果を示す．適用前の変動係数は 0.52 に対して適用後は 0.17 であることから開発組織全体において，設計者，実装者，テスト/評価の担当者の負荷が平準化できていることが分かる．

適用の前後におけるサブシステム毎の開発工数の比較結果を図 12，13 に示す．適用前では，Model の開発組織の

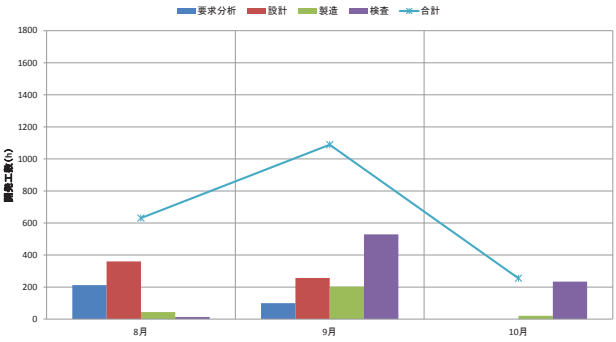

図 9 月毎の工程別の開発工数の推移（適用前）

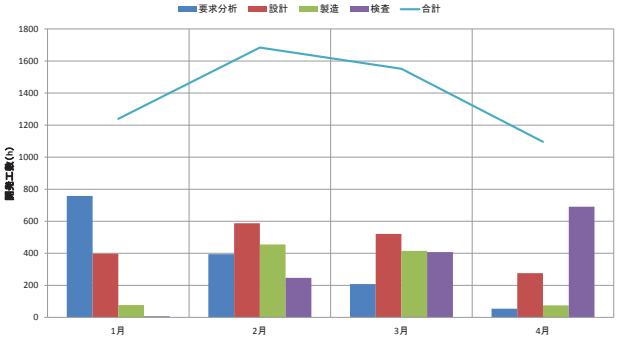

図 10 月毎の工程別の開発工数の推移（適用後）

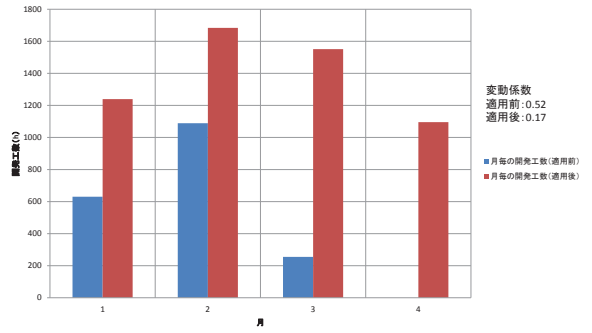

図 11 月毎の開発総数と変動係数の比較

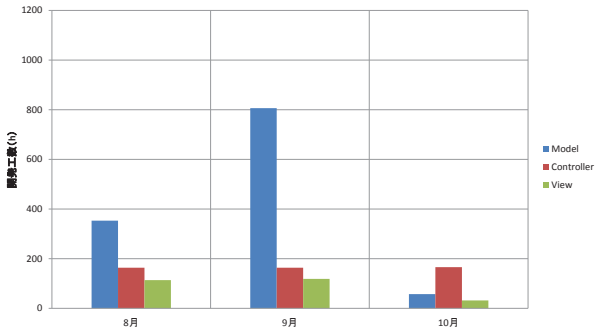

図 12 サブシステム毎の開発工数（適用前）

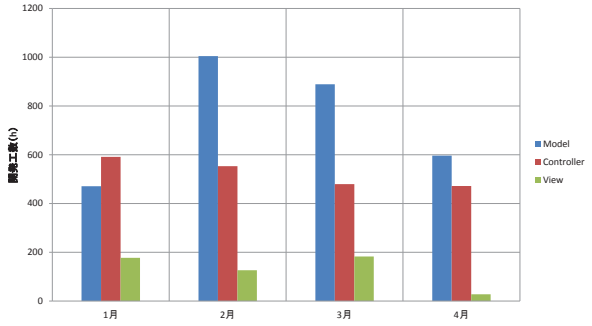

図 13 サブシステム毎の開発工数（適用後）

工数は開発の中盤に集中していることが分かる．一方，適用後では，Model，View，Controller の開発組織の工数は開発の序盤から終盤までほぼ一定であることが分かる．そのため，開発組織内においても負荷が平準化されていることが分かる．以上のことから，開発者の作業待ちが削減された結果，開発効率が向上した．

8. 考察

図 8 に示すバグの内訳を分析するとタイムボックス 9，10 で多くのバグを検出していることが分かる．原因を調査したところ，タイムボックス 9 では，View の単体テストにおいて，それ以前のタイムボックスとは異なり，ユーザビリティなどの観点で単体テストを実施した結果であることが分かった．タイムボックス 10 では，顧客からの急な仕様追加に対応した結果発生したものであった．

以上のことから，開発プロセスの詳細をより厳密に規定し，ユーザビリティなどの観点で単体テストを View の各コンテンツの単体テストと同じタイムボックス中に実施しておけば，ソフトウェア修正の更なる早期化を実現することができ，さらに効果を高める余地があることが分かった．

また，インクリメンタル開発を導入した副次的効果として，BI と異なり開発期間が一定期間で区切られることで，誰が何をいつまでに開発するのかが明確になり，開発チームの開発能力の把握と，開発チーム全体としてのリソース管理がしやすくなった．その結果，タイムボックス 10 で発生した急な仕様追加にも柔軟に対応することができた．BI ではソフトウェアの完成度も開発状況も把握することが困難であり，納期リスクが顕在化する可能性が高い．

一方，BI からインクリメンタル開発に移行した結果として，開発スケジュールの管理や開発メンバーへの周知に対して課題が残った．開発がスケジュール通りに進まない場合には，スケジュールの見直しや人の割当を都度実施する必要があり，開発メンバーとの密なコミュニケーションが欠かせず，管理コストが増加した．管理コストの増加を低減する施策が必要である．

9. まとめ

MVC アーキテクチャの組み込みシステムにアーキテクチャドリブンなインクリメンタル開発を導入することで，バグ検出の早期化と開発チームの負荷の平準化を行うことができた．その結果，納期の達成と品質の確保をすることができた．しかし，管理コストに課題が生じた．

管理コスト増加の課題に対して，プロジェクト管理ツールやデイリースクラムの導入を検討していく．また，メンバーの入れ替わりがあったときでも本開発方法を適用できるよう，ガイドラインやプロセス定義の整備を進め，開発プロセスの標準化を進める．

参考文献

[1] B. W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72.
[2] J. Bosch (ed.), Continuous Software Engineering, Springer, 2014.
[3] D. Leffingwell, SAFe 4.0 Reference Guide: Scaled Agile Framework for Lean Software and Systems Engineering, Addison-Wesley, 2016.
[4] F. Buschmann, et al., A System of Patterns: Pattern-Oriented Software Architecture, Wiley, 1996.
[5] 林 健吾, 他, プロダクトライン開発のための反復型プロセスモデルと管理方法の提案と適用評価, SES 2016 論文集, 情報処理学会, Aug. 2016, pp. 195-202.
[6] 加藤 光治 (監修), カーエレクトロニクス, (上) システム編, 日経 BP, 2014.
[7] F. van der Linden, et al., Software Product Family Evaluation, Proc. of SPLC 2004, LNCS Vol. 3154, Springer, Aug.-Sep. 2004, pp. 110-129.
[8] 前川 直也, 他, わかりやすいアジャイル開発の教科書, SB クリエイティブ, 2013.
[9] R. N. Taylor, et al., Software Architecture, Wiley, 2010.