デジタルプラクティス　Vol.11 No.2(Apr. 2020)

特集号招待論文

アジャイル開発によるMaaSの実現

佐藤義永[1]　仲井雄大[1]　吉田大樹[1]　中山吉浩[1]

[1]（株）デンソー

自動車産業は「100年に1度の転換期」と呼ばれる競争ルールの大きな変換点を迎えており，コネクティッドによるクラウドとの連携や，シェアリングなどの新しい利用形態により魅力的な自動車サービスが多く登場している．これらの技術革新により自動車の概念は大きく変わり，自動車の魅力は単に自動車そのものの魅力だけでなく，自動車に関連するさまざまな活動をより便利にできるサービスの魅力も重要になってきている．したがって，サービス開発では従来の自動車開発以上に顧客のニーズを考慮した開発が求められる．そこで我々はサービスを顧客とともに作るべく，アジャイル開発の導入を図り，現場に適した改善を繰り返し，有効なサービス開発に成功した．本稿ではその主要な実践方法について紹介する．

1. はじめに

自動車における技術進化は，パワートレインやシャーシ，ボデーなどの自動車の基本構成部品，加えてエアコンやモータ，メータなどの電子・電気機器が主である．これにより環境性や安全性，快適性の向上を実現してきた[1]．また，1990年代から情報通信技術を用いた高度道路交通システムとして，カーナビでのVICS情報の活用，ETCによる高速道路料金所の渋滞緩和などが開発され，交通の輸送効率や快適性が向上した．さらに近年では，電気自動車，AI技術による自動運転，新素材による軽量化，コネクティッドやクラウド技術などハードウェアとソフトウェアの両面で進歩が目覚ましい[2]．加えて，自動車や公共交通機関を1つの移動サービスと捉えて利用者の利便性を高めるMobility as a Service （以下，MaaS）が情報通信技術の発達により登場し，既存技術の進化の延長とは異なる技術革新が起きている．

デンソーは既存の基本構成部品や電子・電気機器の技術全般の製品開発，および研究開発を行っている自動車部品メーカである．デンソーにおいても技術革新を率先して実行すべく，電動化，自動運転，コネクティッドカーやMaaSを中心とした技術開発に取り組んでいる[3]．

電動化や自動運転技術は自動車そのものの価値を高めるのに対し，コネクティッドカーやMaaSは車の価値はもちろん，自動車の利用方法にも大きな変化をもたらす．MaaSが解決する課題は，交通サービスの改善だけではなく，物流などのモノの移動やスマートシティの一部として都市の最適化まで多岐にわたると考えられている[4]．MaaSの利用先に応じて顧客も異なってくるため，これまで

以上にサービス提供先の顧客のニーズを汲み取った開発が必要になると考えられる．また，MaaSでは自動車との連携も重要となるため，コネクティッドカー技術の進歩にも合わせた開発が求められる．

MaaS開発における課題を解決するために，デンソーではアジャイル開発によるWebサービスの研究開発を行う部署を新たに設け，MaaSの開発や運用の内製化に取り組んでいる．なお，本稿におけるMaaS開発を，コネクティッドカーから受信したデータを加工・集約し，インターネットサービスとして顧客に提供するシステムとする．本稿では，MaaSの開発を柔軟に進めるためのアジャイル開発の導入検討や実践方法，開発チームの体制について述べる．また，アジャイル開発を用いて開発プロジェクトを進める上で課題となる仕様の決定方法やアプリケーションの開発方法，インフラストラクチャの構築指針についても述べる．

2. アジャイル開発の導入検討

本章では，アジャイル開発の導入検討事例としてMaaS開発へアジャイル開発を導入するに至った理由を述べる．また，複数あるアジャイル開発手法の中からスクラムを選定した理由についても述べる．

2.1 MaaS開発に柔軟に対応するアジャイル開発

MaaSの利用例として，単一のモバイルアプリケーションを介して複数の移動手段を統合的に利用できるサービスがある．MaaSにより利用者は出発地から目的地までの移動手段を一元的なサービスとして捉えることができ，自由に移動手段の選択や予約，決済ができるようになる[5]．加えて，MaaSに付随するサービスとして，移動中の観光案内の発信，飲食店や宿泊先の検索や予約，クーポンの配布など，さまざまなサービスの試みがなされている[6]．

前述のとおり，MaaSは単なる移動手段のサービス化ではなく，移動を起点としたあらゆるサービスに対して応用可能な概念であり，利用用途も多岐に渡る．したがって，MaaSのアイディアは多く出すことができるが，すべてのアイディアが利用者のニーズを的確に満たすとは限らず，利用者のニーズに柔軟に対応したサービス開発が求められる．また，スマートフォンの利用などで容易にMaaS事業に参入することができることから他業種からの事業者の参入が増加してきている．そのため，素早く開発を行い他社に先駆けてサービスを市場に投入することがビジネスの観点で求められる[7]．

このような顧客ニーズが予測しにくい，あるいは社会的，ビジネス的な変化が大きい環境においてアジャイル開発が有効であることが知られており，MaaS開発においてもアジャイル開発が有効であると考え，導入を決定した．

2.2 アジャイル開発の手法選定

アジャイル開発の手法は多く存在しており，代表的なものにエクストリーミング・プログラミング（以下，XP），リーンソフトウェア開発，スクラムがある．いずれの手法も単なるフレームワークであり，どの手法を採用したとしても開発を進められるが，我々はスクラムを採用している．本節ではスクラムを採用した理由について述べる．

アジャイル開発をゼロから導入するにあたり，参考事例が豊富であることが重要である．スクラムはアジャイル開発の手法の中で最も採用実績がある[8]．スクラムガイドと呼ばれる公式のルールブックの中で役割，成果物，プロセスが明確に定義されており[9]，理解しやすいと判断した．加え

て，開発にかかわるすべてを改善していく考え方が，デンソーが持つ小さなカイゼンを日々繰り返す価値観，デンソースピリットのカイゼン[10]，に親和性が高いことも組織に浸透させる上で重要であると考えた．

一方で，前述のスクラムガイドでも述べられているように，習得することは困難である．そこで，スクラムの指導や開発現場で一緒に課題の解決を目指すアジャイルコーチを組織に加えることで，スクラムの導入を素早く行うことができた．

また，スクラムを継続していくためには，単に実践方法だけではなく考え方や価値観を組織に浸透させることが重要である．我々の開発において継続できた理由は以下のとおりである．

- 関係者を巻き込む定期的なイベントの実施
  関係者とスケジュールを調整する際，イベントが定期的であればスムーズに進むことが多い
- 共通する改善マインド
  デンソーには元々，現状に満足せず改善を繰り返す風土があり，改善を繰り返すことで開発力を高める方法論が存在していた．スクラムと共通する価値観が馴染める要因と考えられる．
- 働き方の改善
  チームでタスクを完了させていくため，一部の開発メンバに負担がかかることがない．開発メンバの開発力に合わせたタスク量に調整可能なため，無理な残業が発生しにくい．

3. アジャイル開発を支える組織

アジャイル開発を実践するために開発拠点を新たに設けるとともに，組織も立ち上げた．本章では，スクラムのプラクティスに則りながらも，これまでのデンソーにおけるものづくり文化を継承した開発現場の作り方について述べる．

3.1 アジャイル開発の役割，タスク，成果物

従来のソフトウェア開発の課題として以下が挙げられる．

- プロジェクト関係者が多く合議制のため要件定義が終わらない
- エビデンス重視の品質保証のため膨大なドキュメントを作成する必要がある
- 開発以外の業務（会議，報告書など）が多い
- タスクが属人化している
- 開発者は複数プロジェクトを兼任し，割り込み業務が多い
- 最終工程のテスト工程で不具合が多発し，納期を守るために開発者は残業・休出で対応する．開発者を増員し開発生産性が落ちる

スクラムにはこれら課題を解決するためのルールが含まれている．プロダクトオーナ（以下，PO），スクラムマスタ（以下，SM），デベロッパ（以下，DEV）と呼ばれる3つのロールが定義されており（これを総称してチームと呼ぶ），それぞれに明確な責務が決められている．この責務を効果的に果たすために，我々は以下のような取り組みを行っている．

- POが顧客やプロジェクト関係者の意見を集約し，開発内容や順番などの意思決定を行う．これにより，要件定義のプロセスが単純化でき，プロジェクトとして素早い意思決定が可能となる．一方で，POの判断が製品の魅力に大きな影響を与えるため，POに適した人材として製品が持つべき機能の取捨選択ができ，かつ完成イメージを描けることが重要である．したがって，市場に詳しく，製品に対する明確なビジョンを持つ人材がPOを担当している．言い換えると，単に管理職だからという理由でPOを担当するということは望ましくなく，またIT

知識や開発経験が豊富という理由もPOに適する理由とはならない.

- プロジェクト開始時にPO・SM・DEV全員で，プロダクトのゴールやプロジェクトの価値観を共有する．共有にはアジャイル開発でよく使われる共通理解のツールであるインセプションデッキを用いる．インセプションデッキの作成時にチーム全員でプロジェクトの背景・目的・各方針を検討することで，開発チームの指針の共有を図る．これにより，プロジェクトの成果物としてドキュメントはどの程度必要か，設計やコーディング，テストなど直接開発に関する業務のほかに，どんな作業が発生する可能性があるかを明確にする．なお，開発以外の業務が発生する可能性が初期時点で判明した場合には，可能な限り当該業務をなくすることができないか検討すべきである.
- DEVはUIやサーバ処理といった技術領域ごとに担当を決めずにPOが順序付けしたプロダクトバックログの順番とおりに開発する．これにより，タスクの属人化を防ぐことができる．もちろんDEVごとの開発経験によって不得意な技術分野はあるが，そのような場合には開発者全員で解決する．具体的には，開発者2人で一緒に実装するペアプログラミングや，より多くの開発者が集まって実装するモブプログラミングを行い，相互レビューや技術の共有化を促進する.
- SMはスクラムのルールの定着を図るとともに開発メンバの障害となるものを取り除く．メンバに対して外部から割り込みがないか，メンバ間でのコミュニケーションは正常に行われているかを確認する．問題があれば，チームに対して問題提起を行い，改善を促す．具体的には，POやDEVが兼務しないようにするのは当然だが，他部署からの問合せの対応で時間が使われていないかを見たり，メンバ間でタスクの相談や進捗報告の中でしっかり発言できているかフォローする.

3.2　アジャイル開発の場づくり

従来の開発現場は大きなフロアに複数プロジェクトが混在することがほとんどであった．このような開発ルームでは下記のような課題があると考えられる.

1．開発者は複数プロジェクトを兼任し，割り込み業務が多い
2．開発以外に拘束時間のある業務が存在する
3．開発以外に拘束時間のない業務が存在する
4．議論が活発に行われない

これらの課題はソフトウェア開発にとって障害となるものである．プロジェクトごとに専用の開発ルームを作ることでこれらの問題を解決することを考えた.

1．複数プロジェクトを兼任している場合，突発的に別プロジェクトの作業が割り込んでくることが考えられる．割り込みが多くなると，計画どおりに開発が進まなくなる可能性が高い．この問題について，1プロジェクト専任かつ専用ルームを作ることで割り込み作業を減らすことができた.
2．開発以外に時間を拘束される業務として会議や電話などがある．これらは決められた時間に会議室や電話の前に拘束される業務として，開発の妨げとなる．専用の開発ルームに会議卓を設置したり関係者の席を置いたりすることで，いつでもその場で会議や連絡を行えるようにし，会議準備や連絡等の時間を減らすことができた.
3．拘束されはしないが，開発以外に時間を取られる作業として，業務として報告書の作成やメールなどがある．これらは実施する時間は決まっていないが，開発以外の業務として存在する．専用の開発ルーム内に設置したホワイトボードおよび付箋にて開発状況や予定を管理し，報告を求めている人がいつでも確認できるような状態にしているため，これらの業務時間を減らすことができた.
4．同じ部屋に別の開発チームが居ると，自分の発言を別チームの人に聞かれたくない．という心理が働き，積極的に発言することができない場合がある．専用の開発ルームでは別の開発チームがいないため，心理的に圧力がかかりにくく活発な議論をしやすい環境であると言える.

5．複数拠点やリモートワークは行わずすべて専用ルームで開発を行っている．理由としては2つあり，1つがタスクの管理を単純化するためであり，もう1つがコミュニケーションギャップを回避するためである．開発ルームだけで開発が閉じればタスクの管理はホワイトボードの付箋だけで済むが，開発ルームの外と共有しようとするとWebで参照可能なツールの導入が必要になってきてしまう．また，タスクの詳細までをツール上に記すことは難しい．これらは同じ場所で作業を行えば会話で解決できると考えており，全員同じ専用ルームで開発を行っていくことを重視している．

4. アジャイル開発プロセス

従来のウォーターフォールな開発では，あらかじめ作るべき機能を明確にし，要件定義や，マイルストーンを設ける．その後，達成に十分なリソースの割り当てるという方法が広く使われている．しかし，MaaSなどの技術変化や世情変化の影響を受けやすい開発では，開発前に決定した要件は開発期間が進むほど変わりやすい．そこで，開発を進めながら要件を固めていく必要がある．本章では開発を進めながら実施していく改善や，顧客を巻き込んでいく価値観，目標設定について述べる．

4.1 改善サイクル

継続的な改善サイクルを回す手法としてPDCAがあり，スクラムも同様の考え方をフレームワークに取り入れている．スクラムの各種イベントとPDCAの相関を**図1**に示す．

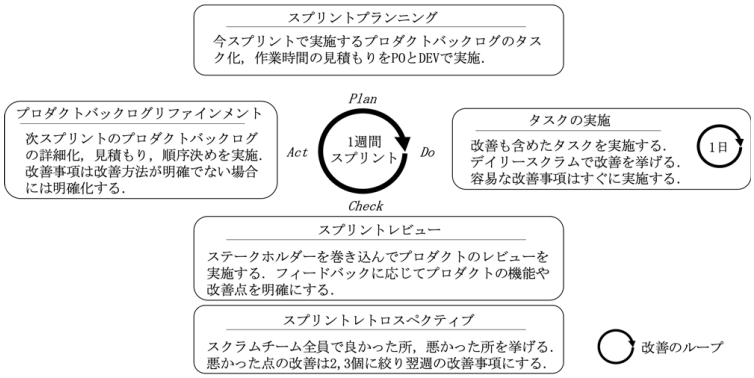

図1　スクラムの改善サイクル

- Plan：スクラムにおける「スプリントプランニング」に該当する．スプリントで構築する機能をPOとDEVで協議の上で決定し，それらを実現するのに必要なタスクの詳細化と作業時間の見積もりを開発者が行い，それを計画とする．
- Do：スプリントプランニングで立てた計画に基づき，実際の開発を進める．途中で計画から外れていないかを「デイリースクラム」で毎日確認する．
- Check：スクラムにおける「スプリントレビュー」「スプリントレトロスペクティブ」に該当する．どちらも開発における改善点の洗い出しであり，「スプリントレビュー」はプロダクトに対して，「スプリントレトロスペクティブ」は開発プロセスに対して改善項目を洗い出す．具体的には，「スプリントレビュー」ではステークホルダーからスプリントの成果物に対して意見をもらい，「レトロスペクティブ」ではチームが自分たちの動きを振り返り，具体的な課題を挙げていく．
- Act：Checkで得たステークホルダーや開発メンバからの意見はPOが必要に応じてプロダクトの価値であるプロダクトバックログに追加し，改善するべき問題点への対策に全員で取り

組む．そこでスクラムにおける「プロダクトバックログリファインメント」によって改善方法の具体化や順序の決定を行う．

以上のように，定期的にPDCAを短いサイクルで回すことにより，継続的に改善を積み重ねる．結果，チームは持続的に成長するとともに自律した組織に変わり，自らプロダクトの価値を高められるようになる．

スクラムは，計画・実行・振り返り・改善のループを回すという点においてはPDCAと同様であると考えられる．

しかし，この中で振り返り・改善に着目すると，レトロスペクティブは下記の点で，PDCAの改善サイクルよりも優れている．

- 一般的にPDCAよりもサイクルが短く，改善頻度が高い
- 個人単位でなく，プロジェクト全体での改善を見込める
- 振り返りに参加する人数が多いため，視点が多様化する
- 計画とおり進めることより，変化に強くなる改善を考える

改善の具体例は以下のとおり．

- 品質基準を満たすための仕組みづくり
  バグを未然に防ぐ仕組みを全員で検討し，手戻りや現場に迷惑をかけることが減った
- ポジティブ，チャレンジ精神の醸成
  失敗したとしても1スプリント期間だけのことなので「まずやってみる」の精神が育つ．結果として，新しいことや高い目標を恐れないアジャイルに適したチームや人が育つ
- チームのモチベーション維持
  成果がスプリントの度に増える，振り返る度にチームも個人も成長していることが実感できる
- 役割を超えた議論により，チームが最適化される
  チーム全員が改善点を出し合うことで，個人でなくチームが変化に柔軟に動けるようになる

改善事例を**図2**と**図3**に示す．図2は開発期間におけるコード品質に問題がある可能性の件数を示しており，件数が少ないほど不具合が発生する可能性やコードの保守性が高いと言える．開発初期段階に件数が高くなっているが，以降は減少傾向にあることが見て取れる．これは，開発初期段階では品質よりも機能実装を優先していたが，レトロスペクティブにおいてコード品質の議論を行い，品質の改善を行ったためである．図3はテストカバレッジである．こちらも同様に開発初期段階でカバレッジが低下したが，品質改善の一環としてカバレッジの基準を80%以上に設定し，さらにサービス開始を見据えて途中から90%に改善している．

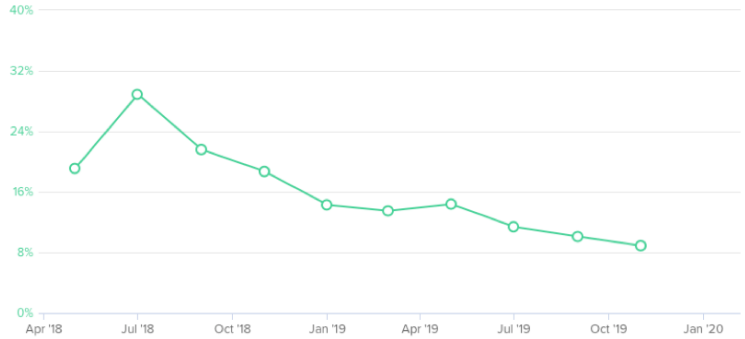

図2　開発期間におけるコード品質に問題がある可能性の件数

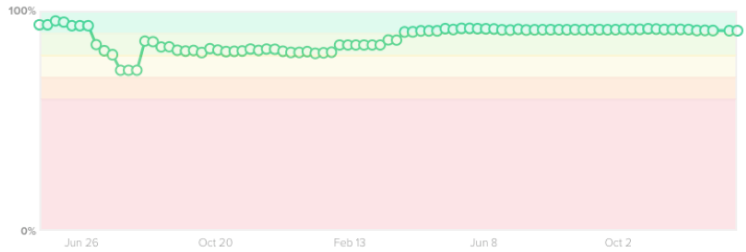

図3　開発期間におけるテストカバレッジ

4.2　顧客の巻き込み

アジャイル開発では関係者全員が価値観を共有することが大事であり，小さい単位で開発・テスト・検証を繰り返す必要がある．そのため，開発チームは開発・テストを素早くできる状態を構築し，維持することに努める．開発チーム以外のステークホルダなどの関係者は，開発された機能を迅速に検証することで開発サイクルをより高速化できる．また，開発チームに限らず，関係者全員が素早く開発から検証までを繰り返すという価値観を持つべきである．

開発者は開発開始時にインセプションデッキにより価値観の共有を行っており，ここでは顧客にアジャイル開発の価値観を共有する方法について述べる．

- 頻繁に開発現場へ足を運んでもらう
  毎週開発ルームで行われるスプリントレビューに，必ずステークホルダ，プロダクトオーナ，開発者が参加し，開発ルームにて成果物に対してレビューを行う．関係者全員が直接意見交換することで，アジャイル開発の価値観を共有することができる．特に顧客の意見に対して開発チームがどのように対応したかを明確にすることが重要である．顧客の意見が取り入れた場合には実際にできあがったモノをお互いに確認することで，また取り入れなかった場合には代替案や取り入れなかった理由は何かをお互い共感することがスプリントレビューを継続し，有意義な場にすることができる．
- 役割を明確にする
  関係者全員がアジャイル開発に取り組んでいることを認識できるよう，誰が何を担当するのか役割を明確にする．顧客を含めたすべての関係者のアジャイル開発における役割を明確にすることで，関係者全員でアジャイル開発の価値観を共有する．

4.3 目標設定

従来の開発では開発すべき全機能を洗い出してからスケジュールに落とし込むが，仕様変更や機能の追加によりスケジュールとおり完了させることが難しくなる．この課題に対し，我々は開発すべき内容のスコープの決定方法の改善に取り組んでいる．

従来の開発ではスコープを固定することが多く，リソースとコストを可変にすることでプロジェクトゴールを目指す．開発すべき機能をすべて作ることを前提としているため機能開発重視と言える．

一方，アジャイル開発ではリソースとコストを固定し，スコープを調整する．これにより開発する機能を減らすこととなるが，その代わりにステークホルダとの会話により機能の開発順序を決めて，順番の高いものから開発する．ウォータフォールと比較すると機能価値重視と言える．なお，アジャイル開発の場合は開発順序が低い機能は開発期間内に完了しないこともあるが，そのような機能はそもそも価値が低いと考えられるため，無駄な開発と言える．

可変のスコープを決定するためには，下記が必要である．

- プロダクトを通じて提供したい価値一覧
  利用者がプロダクトを利用することで得る価値を時系列に並べたユーザストーリマップをチーム全員で作成する．なお，価値の中には非機能要件（セキュリティ，性能，ドキュメント作成など）も含まれる．
- 価値提供のために必要なプロダクトバックログ
  ユーザストーリマップに挙げた価値に基づきプロダクトバックログをチーム全員で列挙する．
- 各プロダクトバックログ項目の開発規模
  開発規模は開発者が見積もる．ある一機能のプロダクトバックログ項目を基準とし，その開発規模をMサイズとする．その他のタスクの開発規模を基準タスクとの相対比較（S，M，L，XL）で見積もる．ただし，本見積は大雑把な見積もりであり，開発規模を保証するものではない．あくまで開発計画が大きく逸脱していないことを確認するためである．
- 価値を提供するために必要な最低限のプロダクトバックログ項目の開発規模
  見積もったプロダクトバックログ項目の内，プロダクトで提供したい最低限の範囲をチーム全員で定義し，その合計規模を最低限の開発規模とする．
- チームが1スプリントで開発する規模
  継続チームであれば以前の規模を参考値とし，新規のチームであれば3スプリント後に実績から算出する．
- リリースのマイルストーン
  顧客と約束している納期や機能がある場合は，チーム全員で共有する．

開発前に初期の準備期間を2週間程度設けている．この期間にインセプションデッキの作成も実施し，プロジェクトの方向性を決めたり，最小限の機能で構成した製品であるMinimum Viable Product（以下，MVP）を定義する．なお，MVPに含める機能は以下の手順で決定する．

1. 関係者全員でユーザストーリマッピングを実施する．
2. ユーザストーリマッピングから得られた実現すべき機能をプロダクトバックログとし，POが開発順序を決定する．
3. DEVはプロダクトバックログに対して開発規模を見積もる．
4. POはプロダクトバックログの開発順序と開発規模，および開発期間からMVPとすべきプロダクトバックログを絞り込む．

開発目標やスコープは開発期間中も市場や顧客ニーズの変化が伴うため，目標設定やスコープ調整は繰り返し行っていくことが望ましい．POは現在のスプリントで開発している機能が目標とおりであるかの確認はもちろん，次スプリントで開発する機能の明確化，以降のスプリントのスコープ調整に注力する必要がある．DEVはPOが正しく状況が判断できるように，機能で不明瞭な点や開発にかかる工数などを随時共有することが重要であり，SMはチーム全体が目標に対して一丸となるように支援していく必要がある．

4.4　開発プロセスのフェーズ

プロジェクトの開発規模に応じて開発期間も長くなることは自明であるが，開発期間が長くなればなるほど市場の変化により仕様変更の可能性が増す．その結果，実装した機能が使われない，あるいは度重なるコード修正で保守性が劣化するといった問題が起こりやすい．

この問題に対して，我々は3カ月程度を目安に開発期間を区切り，4.2節で述べたスコープの決定方針に従って開発機能の絞り込みを実施する．これにより，価値が高い機能の開発が優先して行われ，フィードバックをプロジェクトに反映することが比較的容易となる．また，フェーズ区切りでプログラムのリファクタリングやセキュリティの検査などを実施する目安とできるため，品質の確保にも役立つと考えられる．商用サービスではたとえば次の4フェーズに分割することができる．

立ち上げフェーズ

立ち上げフェーズはプロジェクトの開始時点のフェーズであり，開発対象サービスのMVPを実現するフェーズである．ここでは開発したいサービスの中で最も顧客ニーズが高い部分を実装する．本フェーズでの成果物を顧客に利用してもらうことで，プロジェクトの初期段階で本当に価値のあるサービスであるかの評価が可能である．その結果，プロジェクト後半での仕様変更や，プロジェクトの中止に伴う損失を軽減することができる．

本フェーズの目的はMVPを実現することであり，実装や構築にかかわる作業を少なくしてMVPを素早く実現することが求められる．したがって，実装するアプリケーションの機能は必要最小限とし，インフラストラクチャも最小構成で構築することがよいと考えられる．

巻き込みフェーズ

巻き込みフェーズではさまざまなステークホルダから，より密にフィードバックを得るフェーズである．立ち上げフェーズでも顧客フィードバックを得るが，本フェーズでは機能の追加，非機能要件の決定も含めたサービス仕様を具体化するフェーズである．そこで，スプリントレビューにステークホルダが参加する機会を増やして多くのフィードバックをもらう．リファインメントにおいてもフィードバックをより多く反映して実装すべき機能の精査を実施する．

また本フェーズより非機能要件も含めたアプリケーションの実装やインフラストラクチャの構築を実施する必要がある．詳細は第5章で述べる．

トライアルフェーズ

トライアルフェーズでは限られた顧客向けにサービスを試用してもらうフェーズである．機能検証だけでなく，品質の確認として不具合がなく安定稼働するか，利用する上で応答性能は問題がないかなども検証する．加えて，不具合があれば即時対応できる運用体制の構築も本フェーズで行う．

また，本フェーズでは機能開発よりも不具合修正，性能改善などの品質改善を優先した開発を重視する．

本番フェーズ

本番フェーズは実際にサービスを運用するフェーズである．ここではサービス運用はもちろん，継続的な性能改善，機能追加が行われる．したがって，効率的な運用の仕組みや，運用しながらの機能の追加・更新ができるインフラストラクチャの構築が求められる．これらの工夫点については5.2節で述べる．

5. アジャイル開発によるMaaS開発の実践

車載機と連携したサービスを提供するMaaSでは，車載機から受信したデータを入力としてサービスに応用する．データは位置座標や速度，加速度などをミリ秒単位で収集する．また収集したデータは秒単位でサーバに送信してサーバ側に保存するか，あるいはリアルタイム処理で一次加工を行っている．加えて，市場規模によっては車載機が数十万台規模になることもあり，大規模なサービス環境においても非機能要件を満たす時間内で処理を行う必要がある．

本章では，MaaSを実現する上でデンソーが行った各プロセスの工夫や非機能要件の改善事例を紹介する．なお，本事例で対象とする顧客はサービス利用者であるドライバや，サービスを通じて自動車やドライバを見守る管理者を想定している．

5.1 MaaSアプリケーションの開発

立ち上げフェーズ

MVPとして定義した機能の実装を行うフェーズである．機能の価値検証を行うことを目的しているため，機能の作り込みよりも最小限の実装で開発を進める．また，最初はモックアップで実装し，価値検証を進めながら段階的に実装を行うこともある．

また，車載機との疎通確認をプロジェクトの早い段階で行っておく必要がある．既存の車載機を用いる場合はすぐに結合テストができるが，車載機も並行して開発する場合には車載機開発側と連携を取って進めていく必要がある．具体的には，車載機開発のメンバと容易に連絡を取れる手段，たとえばSlack[11]などを用いる，またはお互いのメンバが同じ開発ルームで頻繁に交流するなど，コミュニケーションを増やすことが有効である．

加えて，素早い開発・検証を行えるようにするため，開発初期段階から自動テスト（RSpec[12]），CI/CD（CircleCI[13]）・コード規約・静的解析ツール（rubocop[14]，CodeClimate[15]）などを活用し，プロダクトコードを書く以外の負荷低減と将来の不具合抑止に努める．

巻き込みフェーズ

立ち上げフェーズでモックアップだった機能を実際に動作するよう実現することや画面デザインを作りこみ，実現したいシステム像に近づける．これによりステークホルダからより現実的なフィードバックを得られると考えられる．

また，フィードバッグによってはこれまでになかった機能の追加や大幅な機能変更が発生する場合がある．これにより開発期間が大幅に伸びる可能性がある．このような場合には，フィードバッグを実現する方法としてコードの変更以外の方法を選択することも含めて検討する必要がある．たとえば，UIの小規模な変更や，既存データの見せ方によって変える，そもそもフィードバッグの内容が本当に必要なものかを検討する．

トライアルフェーズ

本番フェーズを見据えた運用体制を組み，開発チームで機能開発と保守運用を行う．試用中に異常や不具合等が発生した場合，開発チームにて解析を行い，対策の考慮および修正を行う．保守運用は突発的かつ対処完了までの時間が短いことがあるため，機能開発とバランスをとりながら行う．問題発生時の解決策は，原則として暫定対応は行わない．としている．理由としては，恒久対策を実施しない場合，暫定対応が原因で別の問題に波及していくなど，問題が広がっていく可能性があるためである．やむを得ず暫定対応とする場合は，恒久対策の決定と適用時期・タイミングを整理してから行うこととしている．

性能検証においては，期待する性能が得られなかった場合に性能改善を実施するが，アプリケーションの修正が本当に必要であるかの検討も重要である．クラウドプラットフォームを利用している場合には，アプリケーションの変更よりもインフラストラクチャのスケールアップやスケールアウトの方が実現は容易である．したがって，性能改善が必要な場合にはインフラストラクチャの変更のみで対応可能なのかどうか，アプリケーションの修正も必要なのか，という視点で長期的な解決方法の検討を行う．

本番フェーズ

サービスの運用保守，および顧客からのフィードバックに応じた追加機能の開発を行う．どちらも同じ開発チームが対応する．開発内容については，POが機能開発および保守運用の優先度およびバランスを決定する．それぞれの内容について以下に述べる．

- 保守運用
  システム監視（リソースやログ等）や本番フェーズで発生した不具合や性能改善等の対応を行う．また，ミドルウェアやライブラリ等のバージョンアップ対応も含まれる．
- 追加機能開発
  トライアルフェーズや本番フェーズにて認識した価値に対して機能開発を行う．他フェーズと同様，POが順序付けを行い，その順序に従って実装する．

5.2　MaaSインフラストラクチャの開発

5.1節で述べたとおり，開発フェーズごとに達成すべき目的が異なる．立ち上げフェーズでは小規模かつ迅速にインフラストラクチャを構築し，プロジェクトの進展や顧客の規模に応じてインフラストラクチャも拡大させていく必要がある．そこでクラウドサービスの活用が不可欠である．本事例ではAmazon Web Services （以下，AWS）に構築した例を用いてMaaS開発におけるクラウドサービス利用の有用性や，工夫，改善点について述べる．

5.2.1　フェーズに応じたインフラストラクチャ構築

立ち上げフェーズではサービスが動く最低限の環境があればよい．**図4**に立ち上げフェーズにおけるインフラストラクチャの構成例を示す．アプリケーションを動作させるサーバとしてAmazon EC2（以下，EC2）[16]，サービスに必要なデータを格納する Amazon RDS（以下，RDS）[17]のみでサービスを稼働させることが可能としている．もちろん，DBサーバをEC2上で稼働させることも可能であるが，設定の容易さ，以降のフェーズでの必要性からマネージドサービスであるRDSを最初から導入している．なお，極端な事例を挙げればクラウド上にインフラストラクチャを構築する必要はなく，ローカルPC上でサービスを動かすことでも目的が達成できる場合もあるため，本当に必要な環境は何かをチームで議論することが重要である．

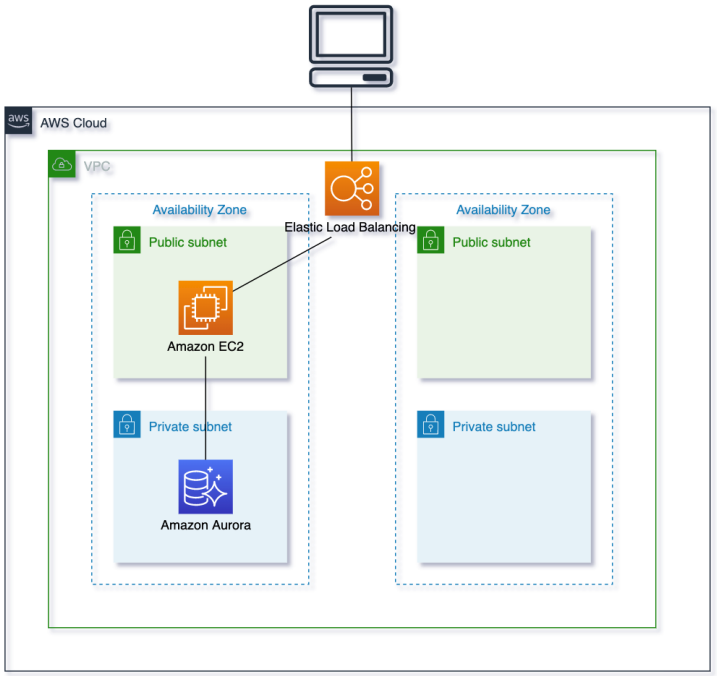

図4　立ち上げフェーズのインフラストラクチャ

顧客巻き込みフェーズにおけるインフラストラクチャの規模は顧客が求める価値，および次のトライアルフェーズまでの期間によって変わる．機能を確認するだけであれば，立ち上げフェーズで構築したインフラストラクチャで十分と考えることができ，一方で応答性能も含めてUXにかかわるのであれば性能を考慮したサーバ性能を用意する必要があり，またさまざまなネットワーク環境から接続する想定があればセキュリティを考慮したネットワークを構築する必要がある．

図5にトライアルフェーズのインフラストラクチャの構成例を示す．本フェーズでは，アプリケーションサーバが稼働するEC2やデータベースサーバであるRDSを冗長化させることで信頼性を高めている．またWebアプリケーションファイアウォールであるAWS WAF[18]やDDoS攻撃を防ぐAWS Shield[19]を導入してセキュリティを堅牢にしている．また，障害に対応できるよう，障害の検出と通知の仕組みも備える必要がある．そこで，Amazon CloudWatch[20]を用いてアプリケーションやマネージドサービスのログ，およびあらかじめ設定した条件に応じたイベント処理を行い，Amazon SNS[21]とAWS Lambda[22]を介してSlackに通知している．Slackには全チームメンバが参加しており，障害の発生状況を即座に把握できるようにしている．

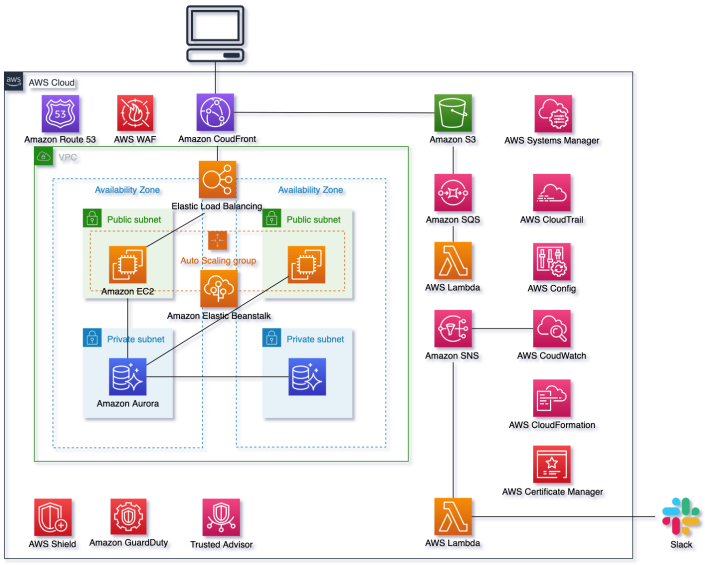

図5　トライアル・本番フェーズのインフラストラクチャ

本番フェーズ主な構成はトライアルフェーズと同等である．異なる点としては，本番サービスの方がサービス利用者数や，扱うデータサイズの規模が大きくなるため，EC2のスペック（CPUやメモリを上げており台数も増える．また，長期運用する上でインフラストラクチャにかかる費用を削減する必要もあるため，AWS Cost Explorer[23]やAWS Budgets[24]を活用して費用を監視している．

5.2.2　インフラストラクチャ構築の自動化

インフラストラクチャは複数環境作成する必要がある．具体的には，開発用環境，ステージング環境，本番環境として3環境作られることが一般的である．また，利用者の数や属性に応じて本番環境を複数作成することもある．したがって，インフラストラクチャの構築は運用に含まれる．すでに構築済みのインフラストラクチャと同等の環境を作成しようとしても，構築に時間がかかる，構築時にミスをしてしまい同じ環境を構築できないといった問題が起こる場合がある．また，サービス利用者の増加に応じてスケールアウトしたり，サービスの機能追加に伴いインフラストラクチャの構成を変更したりする場合にも，現在のインフラストラクチャの構成を踏まえて変更する必要があるため，作業量は膨大になる．

この問題に対して，マネージドサービスを積極的に利用して構築負荷を減らしている．具体的には，データベースサーバは構築せずにRDSを活用する，ロードバランサもElastic Load Balancingを活用して構築に要する時間や運用にかかる管理を低減させている．また，インフラストラクチャをコード化するInfrastructure as Code（以下，IaC）も有効である．インフラストラクチャの構成をコードで実装することで構築を自動化できるため，少ない手順でインフラストラクチャを構築できるようになる．加えて，インフラストラクチャの変更は必ずコードを介して行うことで，変更の管理や再現性の確認が容易になる．コード化する上で以下の工夫を実施している．

- コード化しない部分の見極め
  構築するインフラストラクチャすべてをコード化したくなるが，コード量が増えるほどコードは複雑に，かつ不具合も増える．その結果メンテナンスコストも増えてしまう．したがってコード量は少ない方が良く，コード化しない部分の見極めが重要である．具体的には初期

に構築してしまえば以降は変更しない個所，DNSなど環境に依存する場所はコード化せずに手動で構築する．

- 変数の外部定義
  インスタンスの性能や台数，アプリケーションに渡すパラメータなども環境によって異なってくる．このような変数はコードに直接書き込まず，外部定義ファイルにして環境ごとに管理する．
- モジュール化
  複数のプロジェクトでコードを流用することで開発効率を高めることができる．コードを機能ごとにモジュール化しておくことで，プロジェクト単位で機能の取捨選択が容易にできるようになる．
- 有識者レビュー
  コード化することで容易にインフラストラクチャを構築することができる一方で，実装のミスによりインフラストラクチャに予期せぬ不具合を生じさせることもある．コードの実装は各プロジェクトチームのメンバが行い，コードレビューメンバとしてコード化に詳しいメンバを有識者として加えている．レビューの観点としては，バグや設定ミス，特にセキュリティや拡張性に問題はないかの確認を行う．加えて，定期的にコード化トレーニングを有識者からプロジェクトメンバに対して実施し，スキルの底上げを行っている．

6. おわりに

本稿ではデンソーにおける開発事例に基づき，アジャイル開発がMaaS開発に有用であることを示した．

アジャイル開発手法としてスクラムを選択し，アジャイルコーチ指導の下でスムーズな導入を行った．加えて，スクラムのフレームワークが持つ全員で仕事を完成させていくという価値観や改善のマインドがスクラムの定着，および継続につなげることができた．

アジャイル開発は単にMaaSの開発力を高めるだけでなく，価値観を関係者と共有することにより，価値が高い機能を素早く開発することにも向いている．したがって，開発チームはもちろん，関係者とも価値観を共有することが重要である．その取り組みとして同じ場で開発を行うことや各々の役割を定義することが重要である．

アプリケーションやインフラストラクチャの実装では最初から全機能の作り込みを行うのではなく，フェーズに応じて最小限の実装をすることが重要である．開発途中での機能の変更や追加に柔軟に対応するために，初期段階からインフラストラクチャの自動化やコード化も必須である．

我々が開発したサービスにはすでに運用を開始しているものも多く，さらなる規模拡大や機能拡張も予定している．加えて，新たなサービスの立ち上げも予定しており，より効果的に開発を進めていく必要がある．特にサービスの対象台数や対象顧客，サービスのグローバル化といった開発規模拡大が予想され，拡大に応じて新たな課題が出てくる．これまで得たアジャイル開発の知見を活かしつつもさらなる継続的な改善を実施し，MaaS開発の発展に貢献していきたい．

謝辞　本稿の作成にご協力いただいた皆様に深謝いたします．

参考文献

1）寺谷達夫，大熊　繁：電気が進化させる自動車技術，電気学会論文誌D（産業応用部門誌），125巻，10号，pp.887-894 (2005)．
2）松島憲之：大転換期に入った自動車業界～生き残りの鍵は非連続イノベーションへの対応と自己変革～，9月号，月刊資本市場，No.397, pp.4-13 (2018)．

3）DENSO：MaaSの取り組み，https://www.denso.com/jp/ja/innovation/technology/maas/
4）石村尚也，高柿松之介，新川貴士，吉田奈々絵：続・MaaS（Mobility as a Service）の現状と展望～「日本版MaaS」の実現に向けて～，日本政策投資銀行, 調査研究レポート, 今月のトピックスNo.301-1 (2019).
5）Paper, W. and Alliance, M.：https://maas-alliance.eu/wp-content/uploads/sites/7/2017/09/MaaS-WhitePaper_final_040917-2.pdf (Sep. 4, 2017)
6）林　正尚：MaaSを巡る国内の動向，国土交通政策研究所報, 第71号 (2019).
7）金子雄一郎：MaaSへの期待と課題，運輸政策研究, Vol.21 (2019).
8）13th Annual State Of Agile Report：https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508 (2019)
9）Scrum Guide：https://scrumguides.org/
10）デンソースピリット：https://www.denso.com/jp/ja/about-us/philosophy-and-vision/denso-spirit/
11）Slack：https://slack.com
12）RSpec：https://github.com/rspec/rspec-rails
13）CircleCI：https://circleci.com
14）Rubocop：https://github.com/rubocop-hq/rubocop
15）CodeClimate：https://codeclimate.com/
16）Amazon EC2：https://aws.amazon.com/jp/ec2/
17）Amazon RDS：https://aws.amazon.com/jp/rds/
18）AWS WAF：https://aws.amazon.com/jp/waf/
19）AWS Shield：https://aws.amazon.com/jp/shield/
20）Amazon CloudWatch：https://aws.amazon.com/jp/cloudwatch/
21）Amazon SNS：https://aws.amazon.com/jp/sns/
22）AWS Lambda：https://aws.amazon.com/jp/lambda/
23）AWS Cost Explorer：https://aws.amazon.com/jp/aws-cost-management/aws-cost-explorer/
24）AWS Budgets：https://aws.amazon.com/jp/aws-cost-management/aws-budgets/

佐藤 義永（非会員）yoshiei.sato.j7j@jp.denso.com
2012年東北大学大学院情報科学研究科博士課程修了．博士（情報科学）．総合電機メーカを経て2017年より（株）デンソーに入社．Webサービスのアプリケーション開発に従事．

吉田 大樹（非会員）hiroki.yoshida.j3w@jp.denso.com
2010年東京工科大学コンピュータサイエンス学部卒業．同年より，ウォーターフォール開発のサーバサイドエンジニア，フロントサイドエンジニアとしての経験を経て，2019年，（株）デンソー入社．デジタルイノベーション室にてシステムエンジニアとしてアジャイル開発に従事．

仲井 雄大（非会員）yudai.nakai.j3j@jp.denso.com
2009年会津大学コンピュータ理工学部卒業．同年，（株）デンソー入社．ディーゼルエンジン制御開発を経て，2017年デジタルイノベーション室に異動．アジャイル開発に従事．

中山 吉浩（非会員）yoshihiro.nakayama.j8b@jp.denso.com
2005年金沢大学大学院自然科学研究科電子情報科学専攻博士前期課程修了．同年，（株）デンソー入社．2017年デジタルイノベーション室を立ち上げ，アジャイル開発に従事．

採録決定：2019年11月15日
編集担当：細野 繁（東京工科大学）