平成 25 年度

クラウド技術調査 WG レポート

クラウド時代の
新しいソフトウェア開発の潮流

～米国の Agile/DevOps 適用事例からみる
日本の IT 業界が求められる転換～

平成 26 年 4 月

一般社団法人　情報サービス産業協会

技術強化委員会　技術企画部会

クラウド技術調査 WG　開発プロセスチーム

はじめに

平成 22 年度技術委員会先端技術調査 WG にて「クラウドコンピューティングが情報サービス事業者に与える影響とビジネス拡大に向けての提言」を発表した。その後、「クラウド時代の…」と枕詞がつくほど、情報サービス産業だけでなくクラウドを活用することが当たり前の時代となっている。

一方、近年 IT をベースとしたイノベーションはビジネスニーズをいかに早くキャッチアップし、ソフトウェアを素早く開発し、顧客からのフィードバックを得ることで、より価値の高いソフトウェアを開発することが成功の鍵となっており、ビジネスのイノベーションが盛んな米国を中心にアジャイル開発、DevOps(a portmanteau of development and operations)等の新しい手法が注目され、かつ主流になりつつある。

本調査では、クラウドの利用が当然となっているアジャイル開発を中心としたソフトウェア開発手法を解説するとともに、主に米国のスタートアップ企業における成功事例から得られる教訓を考察する。さらに日本の IT ベンダがクラウドを最大限活用してアジャイル開発などに代表される新しい開発手法にいかに取り組むべきか提言を行う。

平成 26 年 4 月

技術強化委員会　技術企画部会
クラウド技術調査 WG　開発プロセスチーム
(株式会社 NTT データ)
主査　出本　浩

執筆者名簿

座長	吉成　安宏	富士通エフ・アイ・ピー株式会社 テクニカルソリューション統括部長 (兼)ビッグデータ推進室　室員 (兼)HPC サービスビジネス推進室　室員
主査	出本　浩	株式会社 NTT データ 技術開発本部 ALM ソリューションセンタ　課長
委員	佐藤　聖規	株式会社 NTT データ 技術開発本部 ALMソリューションセンタ シニア・エキスパート
委員	古瀬　正浩	株式会社インテック 研究開発部
委員	宮川　勉	株式会社 YCC 情報システム 設計部公団体システム課
事務局	鈴木　律郎	一般社団法人情報サービス産業協会 企画調査部　次長

目次

第1章
クラウドコンピューティングのトレンド

1　クラウドコンピューティングのトレンド

1.1　ビジネスを変えるクラウドコンピューティング

IT ベンダにおいて、クラウドをとらえる切り口は 2 つある。

- ✓ ユーザ企業におけるクラウドサービス利用の拡大
- ✓ ソフトウェア開発におけるクラウドコンピューティングの利用

ここでは、ユーザ企業におけるクラウドサービス利用の拡大について記述する。

ここ数年、クラウドサービスは確実に普及している。一時期はクラウドサービスに対する過度な期待が多かったが、現在はクラウドの特徴を理解したうえで、ビジネスにとってどのようなメリットやデメリットがあるかを冷静に判断する時期に入ろうとしている。クラウドサービスが提供され始めた初期の頃は、BtoC 型サービスの急激なアクセス変動や、早急な市民サービス立上げのために適した利用形態と言われていた。しかしながら、プライベートクラウドやハイブリッドクラウドという形態が企業に認知され始めると、企業の基幹システムに対してもクラウドコンピューティングが受け入れられるようになっている。

クラウドコンピューティングが備えるべき特徴は次の 5 つに集約される。[1]

- ➢ 資本、コストや運用、保守の負担がなく、安価で高機能な情報処理能力を利用できる
- ➢ システム構築・開発期間が短縮され、需要変動を柔軟に吸収できる
- ➢ ネットワークを通じた協働、データ収集、実空間制御が容易になる
- ➢ 端末側へのデータ複製・保存が不要となり、端末紛失等による情報漏洩リスクを低減できる
- ➢ クラウド基盤側の冗長性によって、事業継続性が向上する

これらの特徴から企業の利用シーンにおいては、クラウドコンピューティングはスピード、柔軟性、スモールスタート、スケーラビリティを実現する有効な手段として利用されている。

一方、近年のスマートデバイスの普及を反映して、スマートデバイスのビジネス利用も盛んに行われている。端末側へさまざま

なサービスを提供するためにはセンタ側のバックエンドサービスが必須であることから、バックエンドサービスの提供プラットフォームとしてクラウドの重要性が増している。

クラウドサービス市場は着実に拡大している。米IDC(International Data Corporation)の世界のパブリッククラウド市場予測では[1]、2013 年は 474 億ドルで、2017 年には 1,072 億ドルになると予測している。2013 年から 2017 年の年成長率は 23.5%で、IT 産業全体の成長率の 5 倍と予測している(図 1-1)。また、世界のプライベートクラウドインフラ市場は、2012 年に 123 億ドル、2017 年に 222 億ドルと予測している[2]。クラウドサービスの利用内訳は、情報共有、電子メール、ファイル共有が上位を占めるのは日本と同様の傾向であるが、基幹系システムでもクラウドが利用されており、日本との違いがある[3]。

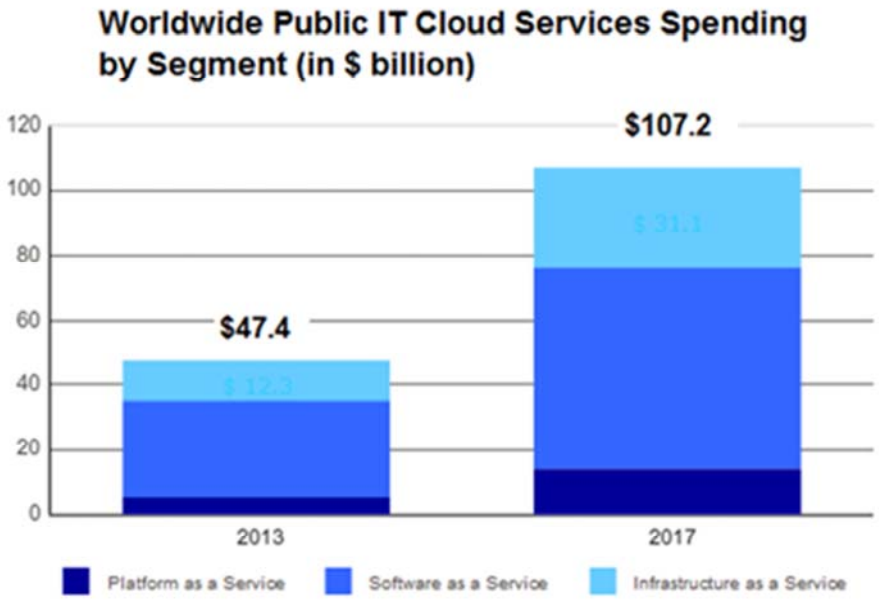


図 1-1　世界のパブリッククラウド市場規模・予測(出典 IDC, 2013)

国内のクラウド市場は、MM 総研[4]によると年平均 32%の成長を続け 2015 年度 1 兆 2558 億円、2017 年度 2 兆 411 億円に達す

[1] https://www.idc.com/getdoc.jsp?containerId=prUS24298013
[2] http://www.idc.com/getdoc.jsp?containerId=240624
[3] http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h25/pdf/n4400000.pdf
[4] http://www.m2ri.jp/newsreleases/main.php?id=010120130828500

ると見込まれている(図 1-2)。

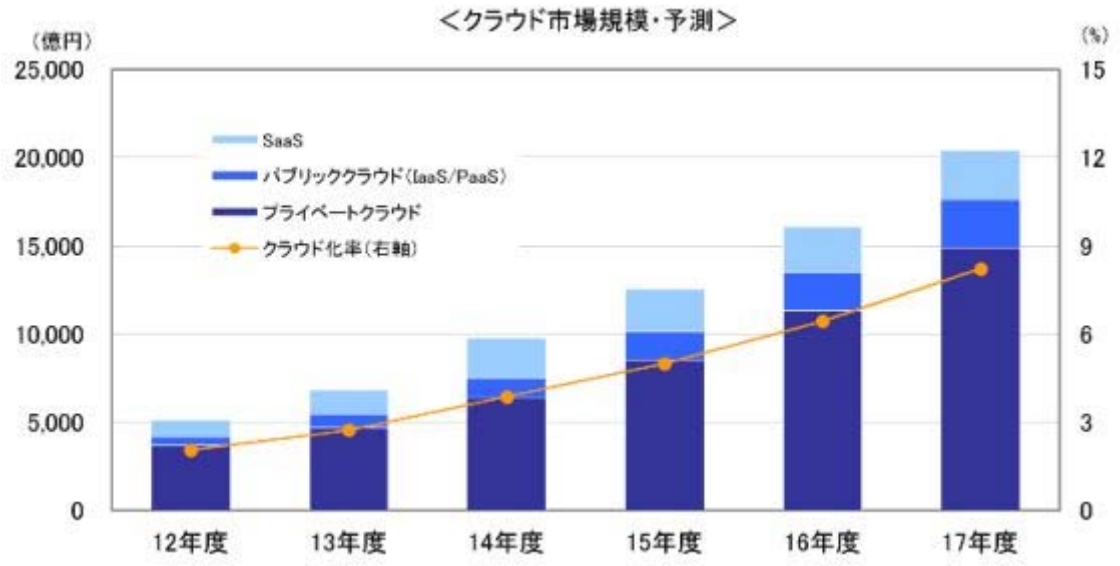


図 1-2 国内のクラウド市場規模・予測(出典 MM 総研, 2013)

なかでも、プライベートクラウドへの投資が 7 割を占めると言われ、コストだけでなくセキュリティを重視する日本企業の特徴が表れている。平成 25 年版情報通信白書によると、クラウドサービスの利用内容としてはファイル保管、データ共有が最も多く、電子メール、サーバ利用が上位を占めている(図 1-3)。[2]

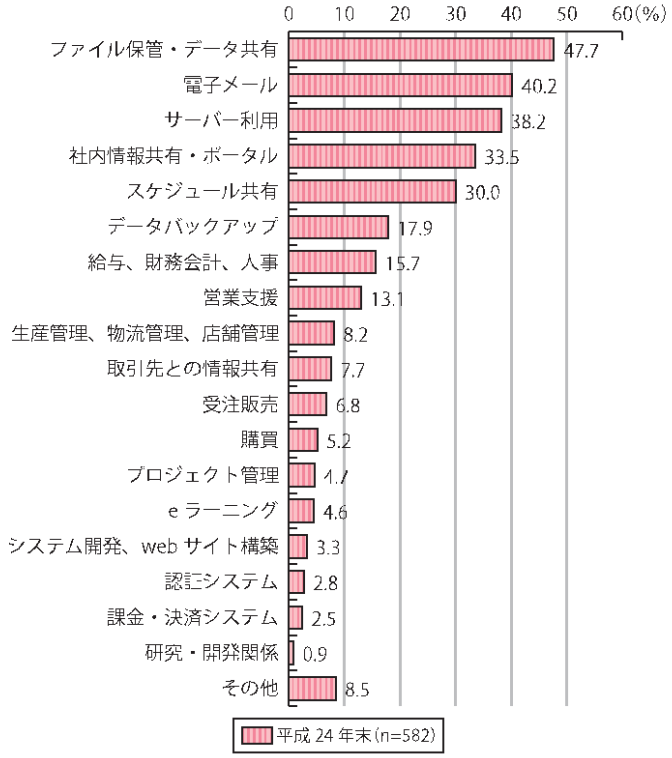


図 1-3 クラウドサービスの利用内訳
(出典 総務省「平成 25 年版情報通信白書」)

企業のクラウドサービス利用動向は、一部でもクラウドサービスを利用している企業は 28.2%あり、前年より 6.6 ポイント増加している。今後利用予定の企業を含めると 48.5%に上る(図 1-4)。多くは、資産を社内に持つ必要がない、初期導入コストが安価、という理由から導入されている(図 1-5)。

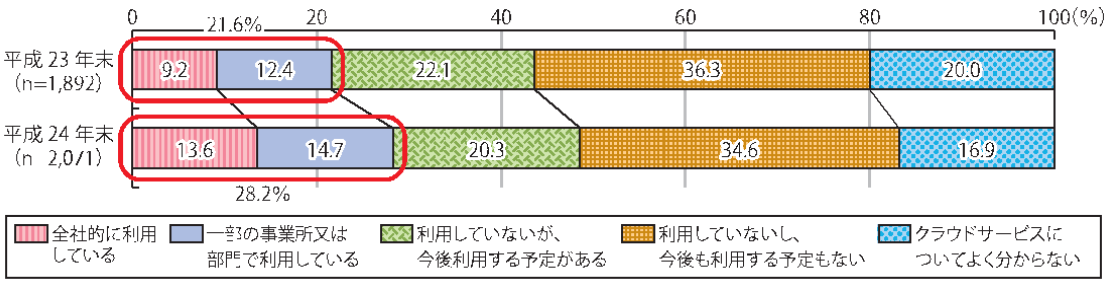


図 1-4 国内におけるクラウドサービスの利用状況
(出典 総務省「平成 25 年版情報通信白書」)

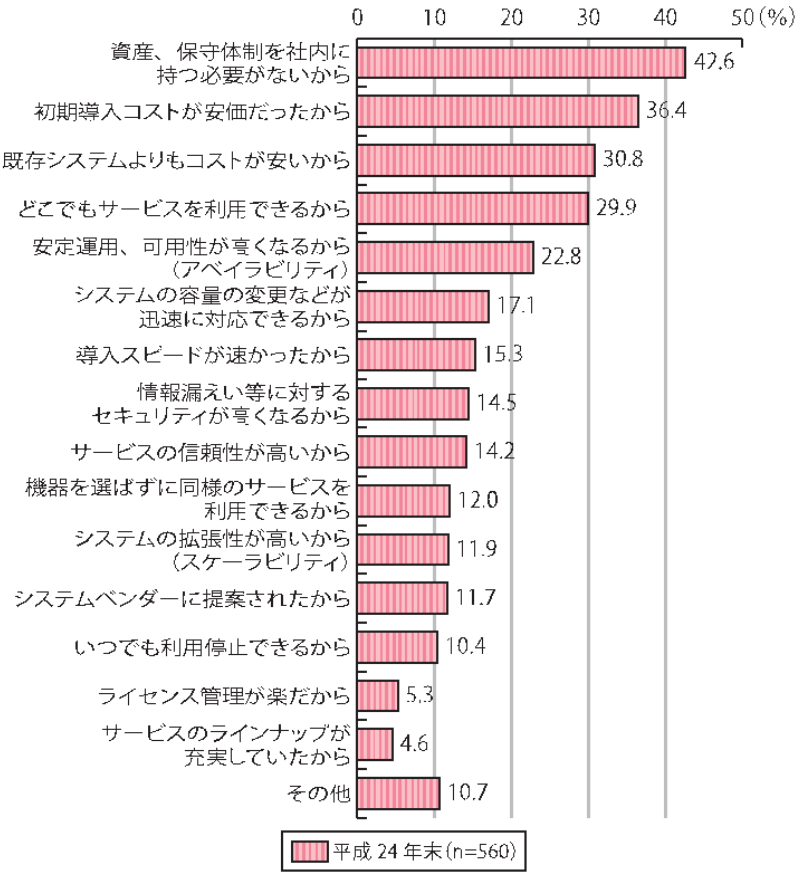


図 1-5　クラウドサービスの導入理由
（出典 総務省「平成 25 年版情報通信白書」）

クラウドサービスを利用しない企業でも、検討した上で利用しない理由が挙げられており、クラウドサービスの理解が進んでいると言える(図 1-6)。利用しない理由としては、必要がない 41.7%に次いで、セキュリティの不安 34.4%、導入の際の改修コスト増 22.8%となっている。セキュリティの不安を取り除くことがクラウドサービス普及の重要な課題である。

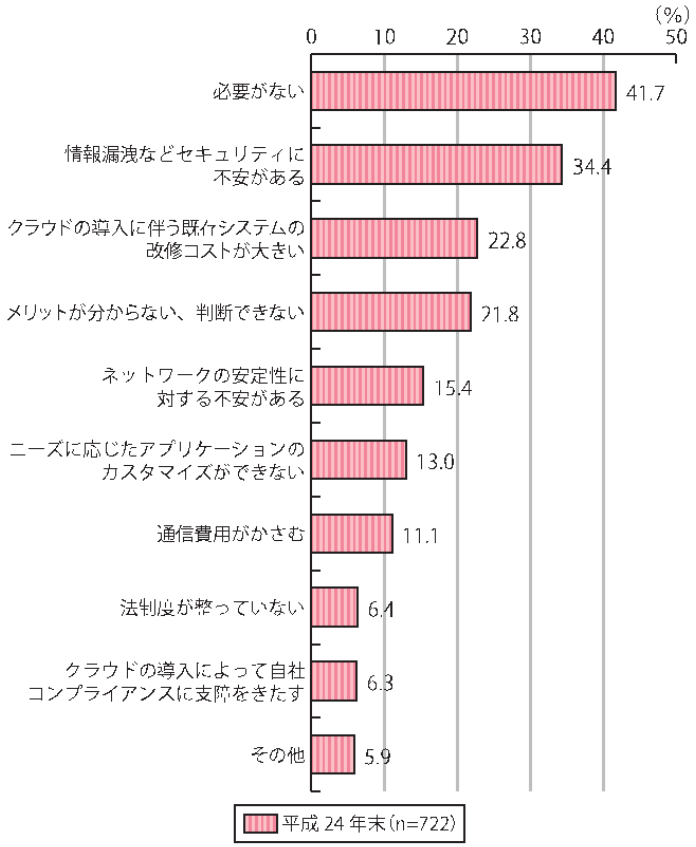


図 1-6　クラウドサービスを導入しない理由
（出典 総務省「平成 25 年版情報通信白書」）

このように、ユーザ企業のクラウドサービス利用は今後も増えるであろう。さらには、システムを構築する際にクラウドサービスの利用をまず検討する「クラウドファースト」が当たり前になりつつある[5]。

クラウドサービスの利用にはメリットだけでなく、リスクも伴う。クラウドサービスを利用すると、災害時のデータ復旧に期待が持てる反面、クラウドの障害により企業のビジネスが停止したり、悪くすると業務データを失う可能性もある。これらのメリットとデメリットを考慮しながら、クラウドサービスの利用は拡大していくと考えられる。

これからのクラウドサービス利用は、利用するかしないかではなく、どうクラウドサービスを利用するかに移ってきている。その一例が、専門分野別クラウドサービスの立ち上がりである。「自治体クラウド」「農業クラウド」「ヘルスケアクラウド」などの業

[5] http://www.m2ri.jp/newsreleases/main.php?id=010120130828500

界クラウド(コミュニティクラウド)は、特定用途に特化することで利用メリットを高めている[6]。

1.2 IT ベンダ企業へのクラウドインパクト

ユーザ企業でのクラウドサービスの普及に伴って、情報システムの位置付けが所有から利用へと変わってきている。システムを委託開発する案件が減少し、既存サービスを組み合わせることで自社の業務システムを構築する形態が増えると言われている。これは IT ベンダにとって SI 案件の市場が縮小することを意味する。IDC Japan[7]によると 、情報システムの構築、導入、運用などを含む国内 IT サービス市場予測は、2017 年の市場規模は 5 兆 3,151 億円とクラウドサービス市場の 2 倍以上であるが、年間平均成長率は 1.5%とクラウド市場に比べてはるかに低成長と予測している(図 1-7)。

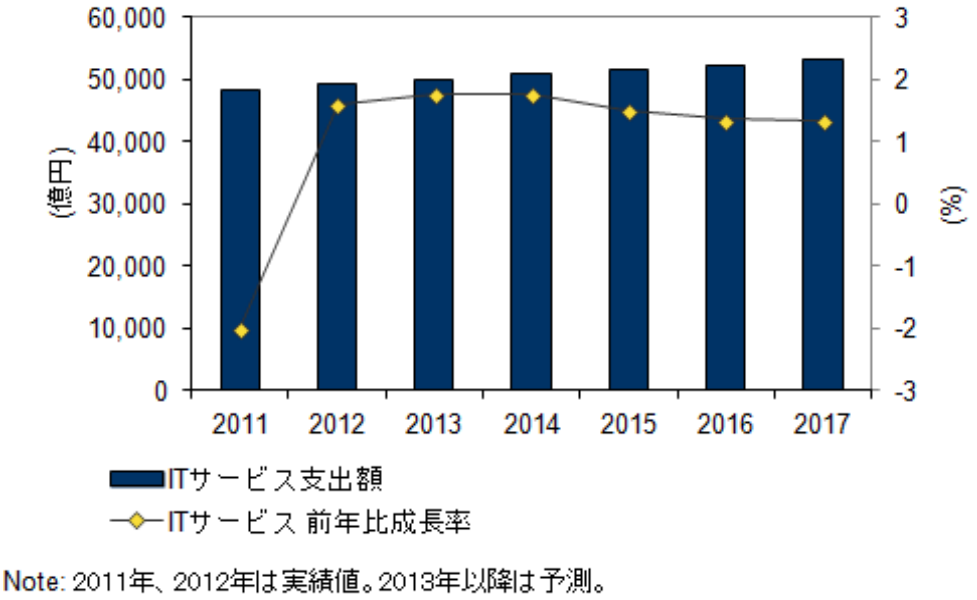


図 1-7 国内 IT サービス市場 支出額予測：2011～2017 年
(出典 IDC Japan, 2013)

委託開発案件が減少するとしても、業務内容によってはクラウドサービスの利用だけではカバーしきれない要求が必ずあり、システム開発が無くなることはない。しかし、従来のシステム開発

[6] https://www.ipa.go.jp/files/000026399.pdf
[7] http://www.idcjapan.co.jp/Press/Current/20131030Apr.html

に比べて、ユーザ企業から IT ベンダ企業に対して以下の要求が高まっている。

- ✓ システム開発コストの削減
- ✓ システム開発の短納期化
- ✓ 仕様変更の柔軟性
- ✓ 保守費からの解放

ユーザ企業の期待に対して、IT ベンダー企業はクラウドを利用したシステム開発によって効果を出そうとしている。たとえばクラウドの特徴を生かして、

- ✓ リソース調達の短縮
- ✓ 開発およびテスト環境の共通化、標準化、充実
- ✓ 分散開発の容易化
- ✓ 保守環境の維持
- ✓ マルチプラットフォーム環境の保持
- ✓ 集中した品質向上サービスの提供
- ✓ 過去の開発資産の再利用

が行われている。ユーザ企業が利用するシステムがクラウド上で稼働するのであれば、システム開発段階からクラウド上で開発を進め、スムースなサービスインが可能になる。またリソースの調達が容易になることから、開発環境を素早く用意できる。さらに開発環境のクラウド化によって副次的な効果も生まれている。それは開発環境が一カ所に集約されることである。リソースの有効利用というだけでなく、開発環境を標準化し、共通の開発支援機能をプロジェクト側にサービスとして提供することが可能になる。具体例として、

- ✓ 共通の開発支援ツールの提供
- ✓ 共同で利用するテスト環境の提供
- ✓ 品質向上のための専門サービスの提供

といったプロジェクトの垣根を超えた利用効果が得られる。

IT ベンダ企業では、クラウドの特徴を活かした開発環境の取り

組みが行われている。それらの取り組みを表 1-1 に示す。

表 1-1 クラウド開発環境に関する IT ベンダ企業の取り組み

NTT DATA[8]	クラウド開発環境を活用して超高速開発を志向。オープン系システム開発のための社内フレームワーク「TERASOLUNA」を2007年以降オープンソースとして公開。
新日鉄住金ソリューションズ[9]	全社システム開発、テスト環境「NSSDCクラウド」を構築。開発プロセスや成果物、プロセス管理ツール群の標準化を行い、設計からテストまで一連の開発工程を司る環境と開発支援を一体でサービス化する。
富士通[10]	アプリケーションフレームワーク「INTARFRM」を社内クラウドサービス化。開発環境のテンプレート提供、開発支援サービスの提供。開発者PCも仮想化の対象としている。
NEC[11]	クラウド型共通ソフトウェア開発環境「ソフトウェアファクトリ」を自社に提供。クラウド上でプロジェクト管理や開発ツールの提供、ソースコードや仕様書の共有、バグ検出などを利用可能にする統合開発環境。東日本と西日本のデータセンタ間にOpenFlow技術を導入して最適経路設定を実施。
SCSK[12]	「PrimeCloud for Developers」。システム開発プロジェクトの管理に必要な機能を実装したプロジェクト管理環境およびサーバリソースをオンデマンドで簡単に調達できる機能を併せ持ったクラウド型開発環境サービス。
日本IBM[13]	「IBM Smarter Cloud Application Services(IBM SCAS)」。アプリケーションの開発環境とアプリケーション・ライフサイクル全体の管理機能を統合し提供するPaaS。SCASの価格はサービスや機能、使用する仮想パターンごとに、ユーザー数や使用時間にもとづいて設定。

[8] http://www.nttdata.com/jp/ja/news/release/2007/112900.html
[9] http://www.ns-sol.co.jp/press/2010/20100413_110000.html
[10] http://img.jp.fujitsu.com/downloads/jp/jmag/vol63-2/paper19.pdf
[11] http://jpn.nec.com/press/201209/20120927_02.html
[12] http://www.scsk.jp/news/2012/press/product/20120912.html
[13] http://www-06.ibm.com/jp/press/2013/05/2901.html

システム開発にクラウドを利用することは、システムを作るから使うへの転換が進むことを意味する。これらのメリットを活かすことで、IT ベンダのシステム開発においてもスピード、柔軟性、スモールスタート、スケーラブルを実現し、ひいてはユーザ企業のビジネスに貢献できると考える。

参考文献

[1]情報サービス産業協会, クラウドコンピューティングが情報サービス事業者に与える影響とビジネス拡大に向けての提言, 情報サービス産業協会, p14, 2011 年 8 月
[2]総務省, 平成 25 年版情報通信白書, 2013 年 7 月 16 日,
http://www.soumu.go.jp/johotsusintokei/whitepaper/index.html

第2章

ソフトウェア開発プロセスのトレンド

2　ソフトウェア開発プロセスのトレンド

本章では、ソフトウェア開発プロセスの変遷について、その時代背景とともに紹介する

ソフトウェア開発プロセスとは、ソフトウェア開発に関連する要求定義、設計、コーディングテストといった一連の作業プロセスを指し、それらが計画通りの期間、品質で実行できることを目的としている。

その登場は、米国の W. W.ロイスによって 1970 年に発表された論文「Managing the Development of Large Software Systems」の内容が元になったとされる「ウォーターフォール型」と言われている。登場の背景は、1960 年代後半、開発するシステムの大規模化に伴い、システムが複雑化し、それまでの個人のスキルに頼った開発方法では、システムの品質を維持することが困難となったためである。その後は ICT 技術の進展と時代背景による顧客ニーズの変化とともに、開発プロセスも「プロトタイプ型」、「アジャイル型」と変化してきている。(図 2-1)

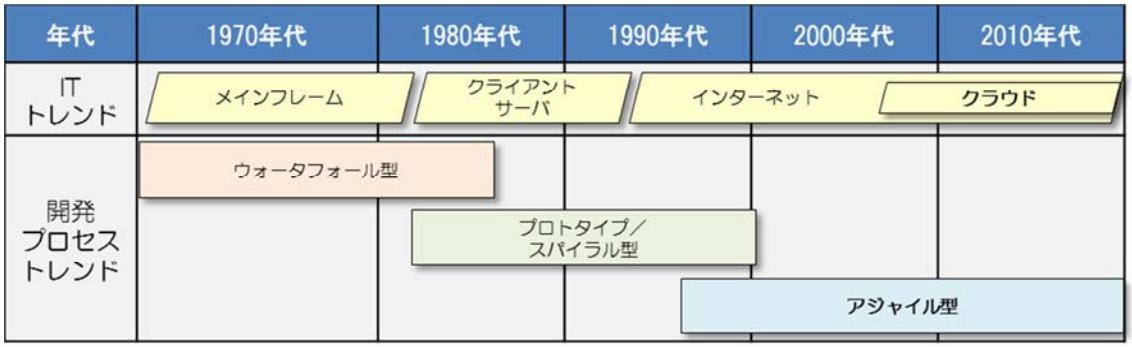


図 2-1 ソフトウェア開発プロセスの変遷

2.1　アジャイル開発とリーン・スタートアップ

アジャイル開発の説明にあたり、世界的にも注目を集めている「リーン・スタートアップ」について解説する。

1990 年代のインターネット時代に入り、消費者のニーズは多様化し、ビジネスサイクルの短縮により、ビジネス環境には不確実性が高まってきた。また過去からの延長線上にない、不連続な変化も起きている。そこで、事業開発に関するコストをかけずに最低限の製品やサービス、試作品により市場の反応を見て改善する、

こうしたサイクルを繰り返すことで、新しい事業の成功率を高めようとするマネジメント手法である。

このリーン・スタートアップとは、エリック・リース著『The Lean Startup』(邦題 『リーン・スタートアップ』)で提唱された。

名前の通り、トヨタ自動車のリーンプロダクションシステムから着想を得た手法であり、素早い事業の立ち上げと成功を目指すため、構築、計測、学習(Build-Measure-Learn)のフィードバックループを行う点に特徴がある。この手法では最初の「製品」に求める要件として「実用最小限の製品(MVP: Minimum Viable Product)」を挙げている。

リーン・スタートアップは、仮説と検証を繰り返す、いわば失敗と改善を重ねることで顧客の期待に応えようという手法であることから、検証のための「製品」はできるだけ時間やコストをかけずに開発しようと考える(図 2-2)。

この「実用最小限の製品(MVP)」を実現するための IT 環境として、アジャイル開発や継続的デリバリー(Continuous Delivery)、クラウドコンピューティングなどが用いられるようになった。

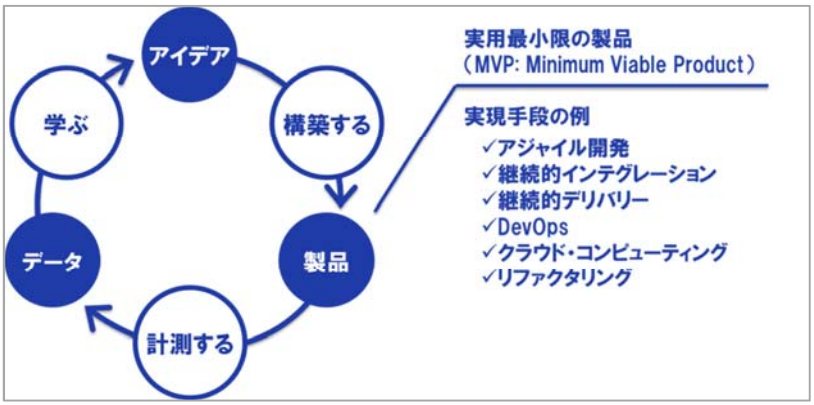


図 2-2 リーンスタートアップとは

2.2 ウォーターフォール型(メインフレーム時代)

メインフレームによる処理集中型のシステムは、1960 年代後半からシステムの規模拡大、複雑化により従来の個人のスキルに頼ってきた開発手法では限界が見えてきた。そのために、ウォーターフォール型プロセスによる開発法が生まれた。

その特徴は、プロジェクトの開始当初に原則としてすべての要

件を定義すること、開発工程を明確に分割し、それぞれの工程の出力(成果物)を次工程の入力とすることである。たとえば要件定義フェーズの成果物である要件定義書は、次の外部設計フェーズでは作業のもとになるドキュメントとして利用される。つまり、各フェーズで明確に定義された成果物を次フェーズの作業メンバが参照する。これにより、属人的な側面を極力減らし、開発すべきシステムの品質を均一化しやすい利点がある。

しかしながら、システムの大規模化が進む中、上流工程で決まった仕様が変更になると、大幅な手戻りが発生する。特にシステム開発プロジェクトの最初から仕様が明確になっていることは、実際にはまれであることや、実装やテストといった作業がプロジェクト終盤に集中しているため、設計ミスがあると手戻りのコストが大きくなりやすいといった欠点もある。

2.3 アジャイル型(インターネット時代)

1980 年代に入ると、メインフレーム時代からクライアントサーバ時代に変わり、ネットワーク、オープン化が進み処理も分散化され、メインフレーム時代よりもシステム規模は大規模になり複雑化し、ウォーターフォール型の問題点が露呈するケースが増え始めた。

このため、ウォーターフォール型に代わる開発プロセスが登場し始めた。代表的なものが「プロトタイピング型」である(本稿では、プロトタイプ型開発の説明は省略する)。そしてクライアントサーバ時代も 1990 年代中期からインターネット時代へと移り始めた。

インターネット時代は、技術のオープン化が拡大し、ネットワークも簡単にグローバルに接続されようになった。このことで企業はビジネス市場を簡単に拡大できるようになったが、その反面グローバルから国内市場を攻略される危険にさらされることとなった。

そのため、インターネット時代の情報システムは、タイミングよく市場投入し顧客を獲得することが重要となり、従来のような大規模なシステム開発を行うのではなく、小さく作り、大きく育てる開発にシフトするようになった。

そういった時代背景の中で、アジャイル型プロセスが正式に誕生したと言えるのは 2001 年 2 月の米ユタ州スノーバードでアジ

ャイルソフトウェア開発手法の分野で著名な 17 名による会議におけるアジャイルソフトウェア開発宣言でのことである。

2.3.1 アジャイル型プロセス(アジャイル開発)概要

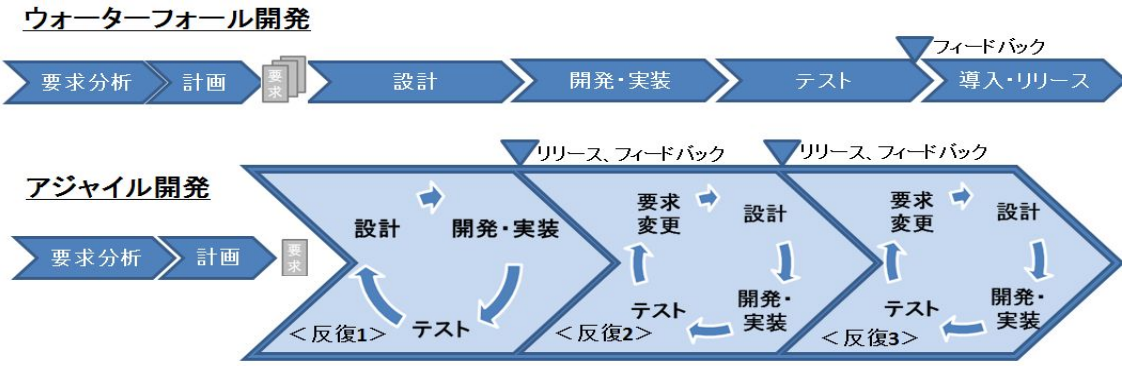
アジャイル開発そのものは、方式ではなく、複数の方式を包括的に説明する用語である。それは、その宣言がスクラム、XP、FDD、Crystal といった方式の提唱者などから構成されていたグループにより行われたことからもわかる。

アジャイルソフトウェア開発宣言では、(従来のウォーターフォール型開発手法と比較して)より良いソフトウェア開発方法として重視すべき以下の 4 点を定めている。

- プロセスやツールよりも個人と対話
- 包括的な文書よりも動くソフトウェア
- 契約交渉よりも顧客との協調
- 計画に従うことよりも変化への対応

抽象的な概念としてはこのように表現されるアジャイル開発であるが、最大の利点は、ビジネス環境の変化に合わせてシステム要求を迅速かつ柔軟に変更できることである。このことから、システム開発者はユーザと密接に連携しながらユーザの要求変更を踏まえて開発を進めることができ、開発工程の後期に仕様変更が生じた場合に膨大な修正作業が発生する、といったリスクを軽減し、ユーザの理想に近いソフトウェアを実現できる。

特に、アジャイル開発モデルでは、システム開発の初期段階から機能テストを実行できるため、ウォーターフォール開発手法のように、開発の終盤になって初めて実施されるシステムテストで工程の完全性を担保できずに重大な動作エラーが判明した場合のリスクを軽減できる。このことから現実的なソフトウェア開発環境に即した高品質なソフトウェアを効率的に開発できると考えられている(図 2-3)。



※ウォーターフォール開発とアジャイル開発の特徴比較

	開発期間	前提にある考え方	要求規定	利点	欠点
ウォーターフォール開発	長期的	工程および ドキュメント成果物	事前(設計前)にすべてのシステム機能における要求を明確に規定	各開発工程における活動の管理やシステム開発予算の見積りが比較的容易	ユーザがシステムの動作を確認できるようになるまでに時間がかかり、要求変更に柔軟に対応できない
アジャイル開発	短期的	変化への対応と ユーザ(顧客)との対話	反復のサイクル毎に必要に応じて要求の優先順位を明確にしながら確認	正しく動作するシステムを早期にユーザが確認でき、要求変更にも柔軟に対応可能	ビジネス価値を創造できる要求規定が難しく、「木を見て森を見ない」開発となるリスクがある

図 2-3 ウォータフォール開発とアジャイル開発比較

2.3.2 アジャイル開発の現状

アジャイル開発の現状について、一般公開されている報告より抜粋し紹介する。

(1) 米国 Visionone 社によるアンケート調査

アジャイル関連製品ベンダーである米国Visionone社によるアンケート調査
アンケート実施期間：2013 年 8 月 4 日～10 月 16 日
回答者数：3,051 名
回答地域：北米 66%、欧州 20%、その他 14%

＜アジャイル開発実績＞

回答者のうち 88%がアジャイル開発を実践と回答しており、欧米を中心に適用が進んでいる。Forrester Research 2012 年 1 月のレポートでも、ウォーターフォール開発 13%に対し、35%と約 2.5 倍の実績があり、米国におけるソフトウェア開発においてアジャイル開発は、ほぼデファクトスタンダードと言える。

＜アジャイル開発手法＞

アジャイル開発手法については、ほぼ「スクラム」(以下 Scrum をスクラムと表記)に集約されてきており、2 番目のスクラム+XP の 11%のおよそ 5 倍である。昨年の報告からは「スクラム」は 2 ポイント向上し、2 位は 3 ポイント低下といった状況である。(図 2-4)

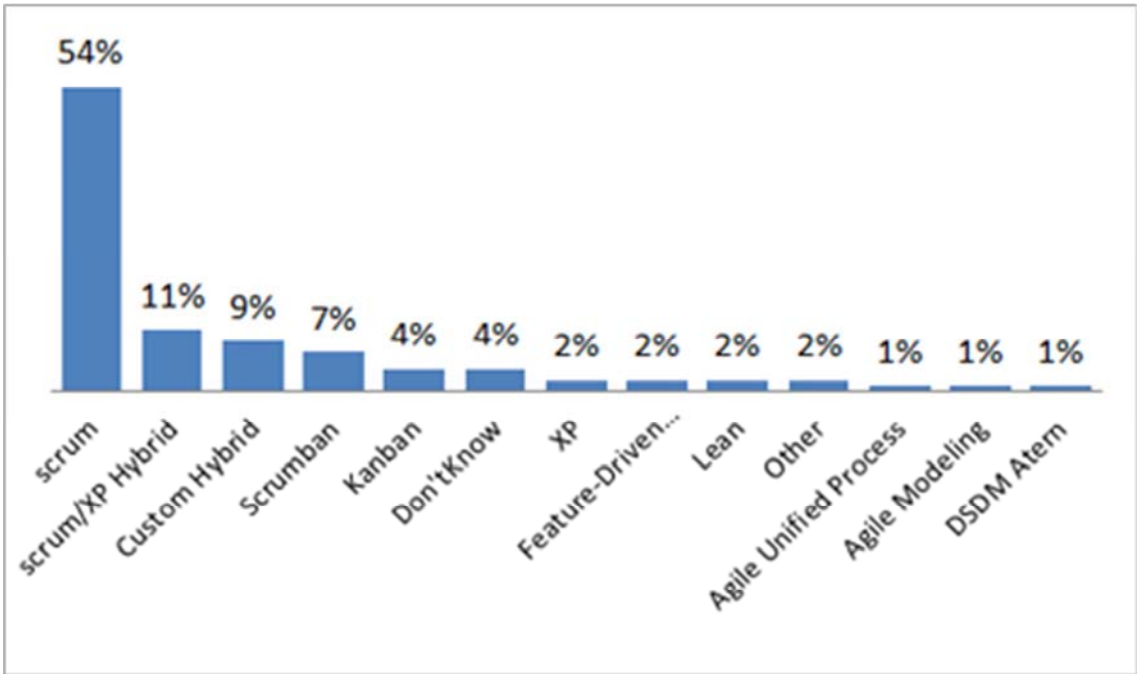
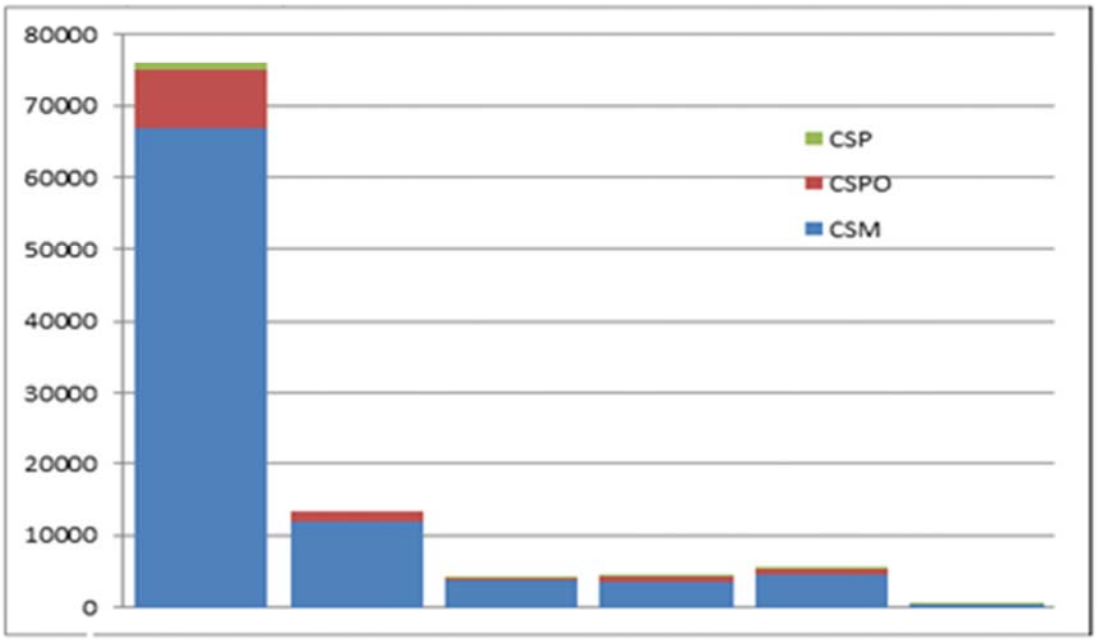


図 2-4　利用アジャイル手法(出典 Visionone, 2013)

(2) 主要グローバル各国におけるスクラム人材の状況

アジャイル開発で多くの人が採用するスクラムの普及状況は、スクラムの認定技術者数から知ることもできる。2012 年(独)情報処理推進機構(IPA)から報告された「非ウォーターフォール型開発の普及要因と適用領域の拡大に関する調査」では、スクラム Alliance 認定資格国別保有者数が報告されており、日本は諸外国と比較して極端に少ない状況であることがわかる(図 2-5)。



	米国	英国	中国	デンマーク	ブラジル	日本
CSM	67000	11800	3800	3700	4600	350
CSPO	8000	1800	400	750	900	120
CSP	1100	0	30	30	60	3

図 2-5　スクラム認定資格保有者数(出典 IPA, 2012)

2.3.3　アジャイル開発における代表的な方式

アジャイル開発は複数の方式を包括的に説明する用語である。ここでは、代表的な方式の概要を紹介する。

(1) スクラム：Scrum

提唱者: ケン・シュウェイバー、ジェフ・サザーランド

現在アジャイル開発で最も採用されている方式である。

スクラムは、複雑なプロダクトを開発、維持するためのフレームワークである。プロダクトを構築するプロセスや技法ではなく、さまざまなプロセスや技法を取り入れることのできるフレームワークである。

スクラムフレームワークは、スクラムチームとその役割、イベント、成果物、ルールで構成されている。

(a) スクラムチーム

スクラムチームは、プロダクトオーナー、開発チーム、スクラ

ムマスターで構成される。

プロダクトオーナー	開発チームの作業とプロダクトの価値の最大化に責任を持つ
開発チーム	プロダクトの開発作業を担当する。チームは、開発、テスト、品質等実装に関する人員をすべて含み、要員は、5～9 名で構成する
スクラムマスター	スクラムチームのメンバーが成果を上げるために支援、奉仕するリーダー

(b) 成果物

成果物は、プロダクトバックログ、スプリントバックログ、インクリメントの 3 つからなる。

プロダクトバックログ	開発すべきプロダクトに必要な機能がすべて並べられた一覧
スプリントバックログ	スプリント期間中に行うタスクリスト
インクリメント	スプリントで完成された機能で出荷判定が可能なプロダクト

(c) 主なイベント

スプリント	開発の反復単位。30 日以内に設定する。
スプリント プランニング	スプリント内で行う開発内容を決める会議
デイリースクラム	毎日行われるミーティング。朝会として 15 分程度開催する
スプリントレビュー	スプリントの最後に行うインクリメントのレビュー
レトロスペクティブ	スプリントの最後に行う改善活動。次のスプリントに反映させる

なお、スクラムの開発者により執筆されたスクラムガイドはダウンロードできる[1]。スクラムの利用を検討されている方は、一読を薦める。

[1] https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-JA.pdf#zoom=100

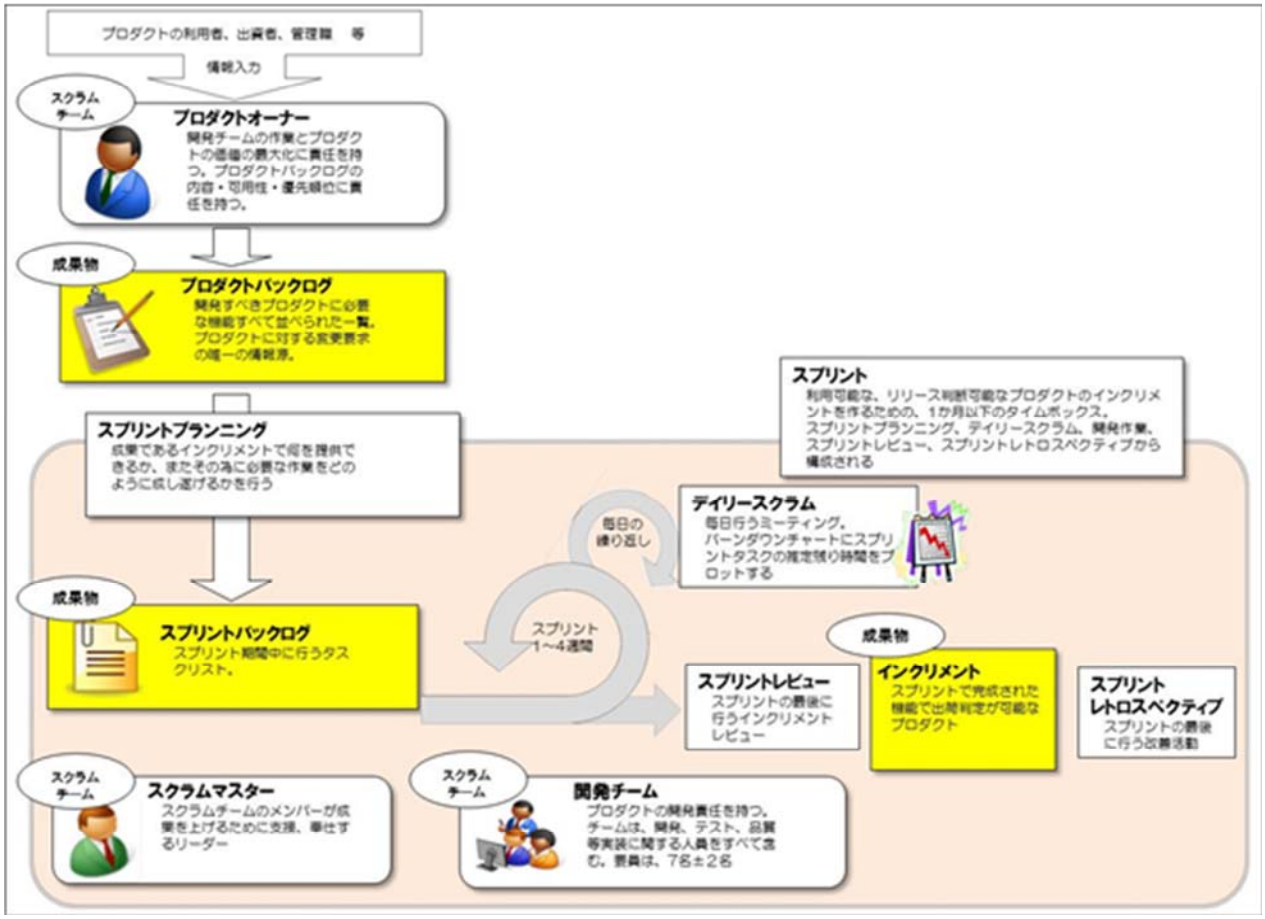


図 2-6　スクラム開発の流れ

(2) エクストリーム・プログラミング：XP (Extreme Programming)

提唱者:ケント・ベック

XP は Extreme Programming の略で、開発に重点を置く内容を定義しており、比較的少人数開発に向いていると言われている。

XP では 10 個程度の具体的なプラクティス(実践)を定義し、ドキュメント整備よりもソースコードを重んじるとともに、開発者一人一人の責任を重視する。

代表的なプラクティスは以下の通り。

(a) 反復

開発は 1～4 週間程度の短い期間に区切り、期間ごとに部分的な設計、実装、テストを行い半完成システムのリリースを繰り返す。

(b) 比喩(メタファ)

システム全体や機能のイメージを何かに例え、ユーザと開発者

とがイメージを共有する。

(c) テスト駆動開発

Test Driven Development(TDD)とも呼ばれる。プログラムロジックをコーディングする前に、(自動化された)テストを作成する。プログラムロジックのコーディングは、そのテストをパスすることを目標に行なう。

(d) リファクタリング

完成済コードでも逐次見直し改善する。コードを改善したならば前述のテストを実行し動作を確認する。

(e) ペアプログラミング

二人のプログラマが 1 台のマシンで共同プログラミングを進めることで、二人がコードを見ることでミスを減らす。

(f) コード共有所有

開発者は誰でも自分以外の人が記述したコードを含めて修正が可能である。

(g) 常時結合

単体テストをパスするコードが完成するたびに、他のコードと結合しテストする。

(h) 最適ペース

最適なペースで仕事を行う。 過労は集中力、想像力を失わせ開発効率が落ちる。

(i) 全員同席

ユーザを開発チームに加え、その外の開発メンバーも含めて全員ができるだけ同席して開発を進めることで誤解を減らし時間を節約する。

(j) コーディング標準

プログラマによってコードの表記にばらつきがでないように、コーディング標準に従ってすべてのコードを書く。

(3) クリスタル(ファミリー) : Crystal (family)

提唱者:アリスター・コックバーン

XP と比較し規律が緩い開発手法であり、明確なルールは、「4 カ月以内のインクリメンタルな開発をする」、「各インクリメント後に反省会(レビュー)を開く」の二つである。

(4) **フィーチャ駆動型開発**：FDD (Feature Driven Development)

提唱者:ピーター・コード、ジェフ・デルーカ

モデル中心の古典的な繰り返し型開発プロセスである。その他の方式が比較的中小規模での適用がメインであるのに対し、大規模開発にも適用できる特徴がある。

(5) **動的システム開発方法論**：DSDM (Dynamic Systems Development Method)

提唱者：デイン・フォルナー

RAD(高速アプリケーション開発)をベースとするアジャイル手法であり、プロトタイプを多用する特長がある。

(6) **リーンソフトウェア開発**：LSD (Lean Software Development)

提唱者:トムおよびメアリー・ポッペンディーク

トヨタのカンバン方式の原理である「リーン(無駄のない)開発」の原則をソフトウェア開発に適用した開発手法である。

参考文献

[1]Winston W. Royce, Managing the Development of Large Software Systems, IEEE WESCON, 1970 年 8 月

[2]エリック・リース, 解説：伊藤 穣一(MITメディアラボ所長), 翻訳：井口 耕二, リーン・スタートアップ～ムダのない起業プロセスでイノベーションを生みだす～, 日経BP社, 2012 年 4 月

[3]ディーン・レフィングウェル, 翻訳：玉川憲, 橘高陸夫, 畑秀明, 藤井智弘, 和田洋, 大澤浩二, アジャイル開発の本質とスケールアップ, 翔泳社, 2010 年 2 月

[4]Jonathan Rasmusson, 翻訳：西村直人, 角谷信太郎, 近藤修平, 角掛拓未, アジャイルサムライ, オーム社, 2011 年 7 月

[5]メアリー・ポッペンディーク, トム・ポッペンディーク, 翻訳：平鍋健児, 高嶋優子, 佐野建樹, リーンソフトウエア開発～アジャイル開発を実践する 22 の方法, 日経 BP 社, 2004 年 7 月

[6]平鍋 健児, 野中 郁次郎, アジャイル開発とスクラム 顧客・技術・経営をつなぐ協調的ソフトウェア開発マネジメント, 翔泳社, 2013 年 1 月

第3章

アジャイルな開発に必要な技術・手法

3　アジャイルな開発に必要な技術・手法

ソフトウェア開発にアジャイル手法を導入することは、新たなソフトウェア開発に従事する開発者と、ユーザのためにアプリケーションを問題なく動作させるためにシステム環境を管理するIT 運用者との間の壁を取り除き、開発と運用の連携を深め、システム障害を軽減しようとする試みにつながるものである。アジャイル開発のコンセプトは、1980 年代の欧米の製造業における新たな製品開発に生産者を巻き込むことで、基本設計、詳細設計、量産設計、試作、生産準備など、各種工程を同時並行的に行って製品開発のスピードアップやコストダウンを目指す「コンカレント・マニュファクチャリング(エンジニアリング)(concurrent manufacturing (engineering))」のそれと似ている。IT 分野では、こうしたアプリケーションの開発と運用の連携を推進するための手法として、(1)DevOps と(2)Continuous Delivery という 2 つのコンセプトが発展してきた。

3.1　DevOps および継続的デリバリ(Continuous Delivery)の技術概要

開発(development)と運用(operation)を組み合わせた言葉である「DevOps」は、ベンダによる新たなソフトウェア開発管理手法を意味し、「継続的デリバリ(Continuous Delivery : CD)」はソフトウェアをエンドユーザに継続的に届けることを意味する。

DevOps は、アジャイル開発の発展に伴い実践されるようになった手法の一つである。従来のソフトウェア開発では、開発者が開発マシン(development server)でアプリケーションの開発およびテストや最適化を実施し、アプリケーションの安定性が確認できた段階で、本番マシン(production server)に移行する、その後、IT 運用者がアプリケーションの本番バージョンの維持、管理を行う。

しかし、クラウド環境では、同一のインフラにおいてソフトウェア開発と運用を並行して行えることから、開発バージョンと本番バージョンとの間の壁が取り払われ、開発者と運用者の効果的なコラボレーションにより、ほぼシームレスにソフトウェアのアップデートを行うことが可能となっている。

継続的デリバリ(以下、CD とする)は、ビジネスがどれくらいの

価値を生み出すのか、素早く検証するために、継続的にサービスに機能を追加する。機能を追加したことで、ユーザからのフィードバックや、システムが引き起こす問題を早期に得ることを目的としている。

従来、ソフトウェアがコンパクトディスク形式[1]で提供されていた時代において、ソフトウェアのバージョンアップを頻繁に行うことは、新たなコンパクトディスクの作成と郵送を意味することから、コスト面で非効率であった。しかし、現在のネットワーク化されたクラウドベースの IT 環境では、このような流通に関わる物理的障害が取り除かれたこともあり、毎日(または 1 日数回)ソフトウェアのバージョンアップを行って、ほぼ瞬間的にユーザにアップデート版を配布することが可能となった。そのような現在の状況下では、ソフトウェアのバージョンアップの頻度および範囲を決定する唯一の要素は、開発者側の能力である。

CD は、インテグレーション、コードの自動実装、継続的試験といった分野において、さまざまな課題に直面している。

GE や Red Bull といった大手企業を顧客として、オンラインおよびオフラインでのソーシャルメディアキャンペーン活動を支援しているデジタル設計企業の米 iStrategy Labs のエンジニアリング部門バイスプレジデントを務める Jon Long 氏[2]によると、「CD は大部分の組織の開発部門にとって、現実(reality)というより目標(goal)の一つである。CD は、不完全なパッチやシステム障害を防ぐために本番環境でのリリースプロセスを監視する IT 運用者に深刻な負担を負わせるもので、多数の企業において、アプリケーションが本番環境で不具合を引き起こせば運用者側の責任が問われるため、大抵の場合、組織の IT 運用者は、本番環境に継続的にプログラムをリリースする CD プラクティスの実践を阻む傾向にある」と述べている。

3.2 開発・試験手法

アジャイル開発手法は、短期間で迅速なコード変更を可能とする順応性の高い開発手法であるが、組織の IT 部門における開発、本番環境におけるインフラ面での「インテグレーション(統合)」

[1] 継続的デリバリの CD と区分するために、ここでは、敢えてコンパクトディスクと表記した

[2] http://istrategylabs.com/team/jon-long/

が、その開発を成功させるために重要な鍵となっている。

従来のウォーターフォール開発とアジャイル開発との主要な相違点の一つに、アジャイル開発では、開発に係る各チームメンバの間で継続的にフィードバックとコミュニケーションを図ることが重視されていることが挙げられる。かつて自身の率いる開発部門でアジャイル開発を実践していた Long 氏は、アジャイル開発手法で競争と連携の間で特有の均衡を維持することに留意している。同氏は「開発に係る個々のプログラマは、どれほどうまくコードを書けるかについて誇示したいと考えているが、最終的には、より良いアプリケーションを生み出すためには他のチームメンバからの意見が欠かせないことを学習する」とし、CD プラクティスにおいて欠かせないプロセスである、各チームメンバによる頻繁なコードの作成、発表およびテストの実施を奨励することで、各メンバ間でのこうした理解を深めることの重要性を強調している。

3.2.1 継続的インテグレーション(CI)

継続的インテグレーション(Continuous Integration : CI)は、アジャイル開発手法の一つである Extreme Programming(XP)における実践プラクティスの一つとして端を発し、米アジャイル開発およびコンサルティング企業 ThoughtWorks Studios のチーフサイエンティストを務める Martin Fowler 氏による論文等でその概念が広められた開発プロセス[3]である。

CI は、短期サイクルを繰り返すことで開発すべき機能を実装するアジャイル開発の根幹となるプラクティスの一つで、各開発者の作成したコードを継続的に、ビルド(build)、テスト、結合することで、開発の早い段階で動作するプログラムをリリースすることを目的とするものである。

CI では、コードのテストを頻繁に実施し、プログラムの問題を早期に特定、修正することを基本的な考え方としており、開発チームメンバが、一日に最低一度は、アプリケーションに対するさまざまな変更コードをまとめ、ビルド自動化システムを用いて各コードを動作させることで、システム障害やエラーを引き起こさないかについて確認するという仕組みとなっている。通常は、自動ツールを使用することで各プロセスは自動化されており、変更

[3] http://www.martinfowler.com/articles/continuousIntegration.html

を完了したコードが「mainline」と呼ばれる中央レポジトリに転送後、同レポジトリは、テスト用に新たなビルドを作成するために用いられる仕組みとなっている。

大部分のソフトウェアアプリケーションは、その機能が複雑であるため、実際の CI プロセスには、ビルドの自動化以外にもさまざまな機能が含まれる。綿密にアプリケーションの統合を行うためには、新コードによる障害を特定するだけでなく、新コードが、ウェブサービスやデータベース、外部リソース、(アップストリームおよびダウンストリーム)ソフトウェアコンポーネントへのリンクを維持しているか確認するためのテストが必要である[4]。そのため、CI システムには、そのシステムプラットフォームの一部に用いられているビルド自動化ツールのほか、こうしたテストを各ビルドの自動化プロセス後に実行するアプリケーションレベルの自動テストツールも必要である。

また、開発、本番システム全体のアーキテクチャが標準化されていれば、自動ビルドおよび自動テストをかなり容易に実施できることから、組織の IT 部門は、CI システムが高度に統合されたソフトウェアインフラを搭載していることを確認する必要がある。

しかしこうしたソフトウェアアーキテクチャとインフラの設計には多大な時間とリソースを投じる必要があり、CI および CD プラクティスを実践したいと考える組織が実際に同プラクティスを実践することは容易ではない。なお、iStrategy Labs の Long 氏は、ウェブベースのスタートアップ企業は「ゼロ」からシステムアーキテクチャおよび開発プロセスを設計し、CI プロセスを最適化できるアーキテクチャを構築することが可能であるため、同様のアーキテクチャを実現するために組織内ですでに標準として受け入れられている開発プロセスやシステム変更を強いられる一般的な企業の IT 部門よりは、同プラクティスの実践が容易であると指摘している。

一方で、CI および CD プラクティスの実践を容易にするための一つのプラクティスとして、ソフトウェアを通じてハードウェアを含むすべての開発環境のコンフィギュレーションを管理し、ア

[4] http://dannorth.net/2006/03/22/continuous-build-is-not-continuous-integration/

プリケーションや他のコンポーネントの自動統合を最適化する「コードとしてのインフラストラクチャ(Infrastructure as code[5])」と呼ばれる手法があり、この手法を適用する際には Chef[6]や Puppet[7]といった構成管理自動化ツールが利用される。しかし、ソフトウェアのテストを効率化・自動化するためのアプリケーションツールを開発する米 ZeroTurnaround がソフトウェア開発者を対象に実施した調査によると、各開発者の所属する組織の大部分において、コードとしてのインフラ手法を活用して管理されているインフラは全体の 10%以下にすぎないとの結果が明らかになっている[8]。

図 3-1 は、CI および CD プラクティスにおける自動ツールを活用した機能リリースに関連したコードのビルド、テスト、実装作業工程を示したものである[9]。

[5] Infrastructure as code では、インフラストラクチャのコンフィグレーションがスクリプトやファイルによって記述されバージョン管理されており、その変更が自動的にデータセンタに反映される仕組みとなっている。従来のようにインフラストラクチャの構成管理を手動で行う必要がない。

[6] http://www.opscode.com/chef/

[7] https://puppetlabs.com/

[8] http://zeroturnaround.com/rebellabs/pragmatic-devops-virtualization-provisioning-with-vagrant-chef/2/

[9] http://blog.assembla.com/assemblablog/tabid/12618/bid/92411/Continuous-Delivery-vs-Continuous-Deployment-vs-Continuous-Integration-Wait-huh.aspx

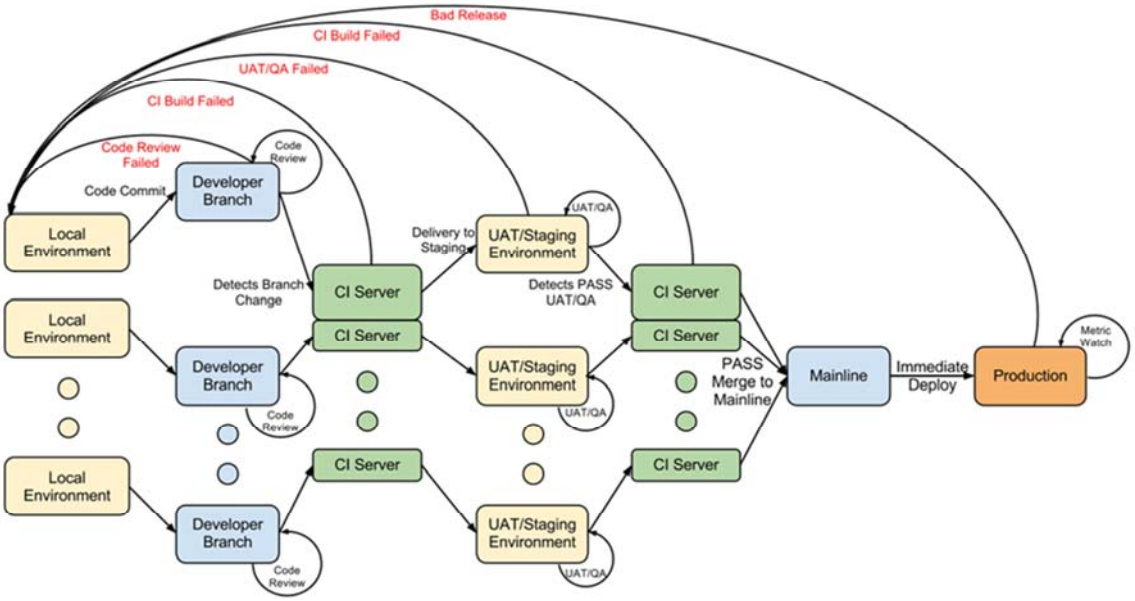


図 3-1 CI および CD プラクティスにおけるコードのビルド、テスト、実装作業工程(出典 Assembla Corporate Blog)

具体的に、その作業工程の流れは、まず、開発者がそれぞれの環境で作成したソースコードが開発用ブランチ(Development Branch)にチェックインされ、各コードは、ビルドテストのために自動ビルドシステム(図 3-1 CI Server)に送られる。ここでビルドに成功した場合、そのアプリケーションは、ユーザ評価テスト(User Assessment Testing：UAT)を行うため、本番環境にアプリケーションが実装される前に本番同様のテストを行うためのステージングサーバ(staging server)に送られる。そして、同サーバで UAT および品質保証(Quality Assurance：QA)テストにパスすれば、自動ビルドシステム(CI Server)において同アプリケーションの当該コードは再コンパイルされ、mainline レポジトリに保存された後、本番環境での実装に送られ、リリースされる仕組みとなっている。組織は、自動ツールを活用することで、CI および CD プラクティスの各作業工程を最初から最後まで自動化することが可能である。

なお、各プロセスでビルドテストや UAT テスト、QA テストにパスしなかった場合、開発者はそれぞれの開発環境に戻って最初から作業工程を実施することになる。図 3-1 に示された開発作業工程はあくまで理想であり、組織のニーズや要求事項、開発方法により、開発工程は多少異なることに留意する必要がある。

3.2.2 テスト駆動開発(TDD)

テスト駆動開発(Test-Driven Development：TDD)は、アジャイル開発および CI プロセスを補完する新たなソフトウェア開発手法として 2003 年に提案され、テスト主導型開発(Test First Development：TFD)から派生した開発手法である。

従来のウォーターフォール型ソフトウェア開発では、最終的なバージョンコードの認証を行うためのテストが、開発プロセスの最終工程で行われるため、多くの場合、より早い段階で解決できた問題が最終段階で発見され、開発者はコードを再作成しなければならない状況に陥っていたことが問題視されていた。

TFD は、同問題に対応するため、アジャイル開発における各開発サイクル(increment)で新たなコードが作成されるたびに複数の簡単なテストを実施し、テストにパスしなければ、その都度デバッグまたは再作成を行って不具合を解消する。そして、同コードがすべてのテストに合格した段階で、開発を継続するという手法である。

TDD は、TFD よりもさらにテスト重視の手法であり、(1)開発者はコード開発を開始する前からテストを作成する必要があり、(2)そのテストに通る最低限のコードを実装後、(3)テストが通る状態を維持しながらコードの重複をなくす等、コード内部の設計を改善するリファクタリングを行う。この(1)～(3)のプロセスを繰り返す手法が TDD である。CI プラクティスを実践する開発者による TDD 手法を採用する利点に関する意見の中には、テストコードが開発プロジェクトの要求事項を規定する文書として機能することから、TDD はソフトウェア仕様を策定するために有用とする意見や、リファクタリングによりソースコードが粗雑になることを防ぎ、開発プロセスにおけるデバッグおよびコードの再作成にかかる手間を軽減できることを主要メリットとする意見が含まれる[10]。

TDD では、各コードに対し、一連の異なるテストを適用しなければならない。一方で、こうしたテストプロセスを手作業でそれぞれ行っていたのでは、開発プロセス全体の遅延につながることから、同テストプロセスを効率化するための自動テストツールを用いる必要がある。現在、こうしたテストツールは多数存在する

[10] http://www.agiledata.org/essays/tdd.html

が、これらは、ユニットテスト(unit test[11])を行ってコードの妥当性を検証するもので、外部データ、サービス、またはソフトウェアコンポーネントへの依存要素をテストするものではない[12]。

3.3 バージョン管理とバイナリ管理

CD プラクティスを実現する上では、ソフトウェアのビルドおよびリリース管理が重要な要素の一つとなっている。

CD では、新たなビルドはほぼ継続的にコンパイルおよび実装されるため、開発されたアプリケーションの正式な「リリース」版は理論的には存在しない。しかし、実際には、組織は本番環境に移されるアプリケーションへの変更機能数に制限を設ける必要があり、特に、大規模な IT 開発部門を持つ組織では、単一のアプリケーションに同時並行で加えられる変更機能が、当該アプリケーションを構成する多数のソースコード(ソースエレメント)に及ぼす影響について追跡する必要がある。

図 3-2 は、インターネット通信販売やオークションを手がける米 eBay において、ある特定の期間に変更されたソースエレメント数(X 軸)と、同エレメント数が機能に占める割合(Y 軸)との関係を示したものである。数百の開発チームを抱える eBay では、アジャイル開発において毎月数百に上る機能を実装しているが、図 3-2 をみると、およそ 40%に上る機能変更が少なくとも 200 以上のソースエレメントに影響するもので、2,000～5,000 ものソースエレメントに影響する機能変更も全体の 5%程度を占めていることが読み取れる[13]。

---

11 ソフトウェアのクラスや関数といった最小構成単位で、ソースコードが対象であれば C 言語等では関数が、Java や C++等ではクラスが一般的にユニットに該当する。

12 http://www.luisatencio.net/2013/03/notes-on-continuous-delivery-continuous.html

13 http://www.ebaytechblog.com/2011/12/15/rapid-development-setup-in-large-environments/

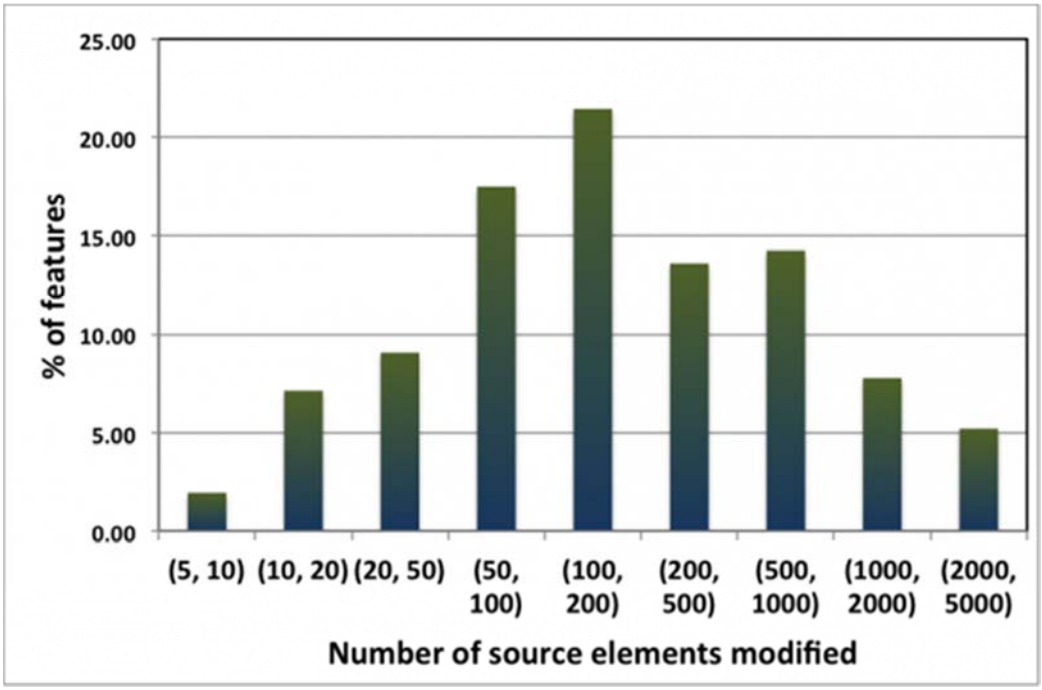


図 3-2　eBay のアジャイル開発における変更機能とソースエレメント数の関係（出典 eBay）

リリース管理は、変更を加えるすべてのコードと各変更に係る依存要素、アプリケーションの特定のビルドまたはリリースに関連した変更箇所を注意深くかつ厳格に記録することで、コード変更の管理に関連した課題に効率的に対応することを目的としている。

3.3.1　Git バージョン管理

多数の IT 組織は、リリース管理を行うために Git バージョン管理レポジトリプラットフォームを採用しつつある。Git は、Subversion、Perforce、ClearCase に類似したオープンソースのソースコード管理(Source Control Management : SCM)システムである[14]。

Git の主要機能の一つとして、たとえば、二人の開発者が特定のアプリケーションにおいて異なる 2 点の変更をサポートするため、同一のコードに同時に変更を加える場合、ソースコードを分岐させて開発の目的や機能ごとに仮の別バージョンを作る「ブランチング(branching)」機能がある。リリース管理ソリューシ

[14] http://git-scm.com/about

ョンを持たない IT 組織は、上の例で、変更した 2 つのコードと、アプリケーション内の他のすべての依存要素に与える影響をそれぞれ個別に分析する必要がある。

一方、Git を用いることで、組織のアプリケーションプロジェクト管理者は、2 人の開発者がそれぞれコードに加えた変更について検討し、(1)いずれの変更も行わない場合、変更前の状態にコードを戻すことや、(2)両方の変更を行う場合、アプリケーションの全体的な安定性を損なうことなく各ソースコードの変更を自動的にマージすることが簡単にできることや、(3)いずれか一方の変更のみを行う場合といった選択肢を含め、各ソースコードへの変更に関する決定を容易に行えるようになる。

Git の主な特徴は以下の通り。

- 1 つのレポジトリにすべての開発者がアクセスする「集中型」と呼ばれるアーキテクチャを採用している Subversion 等のバージョン管理システムでは、1 人の開発者がレポジトリから特定のコードを取り出している(チェックアウトしている)間、他の開発者は当該コードに変更を加えることができない。
  一方、Git は「分散型」のバージョン管理システムであり、各開発者がそれぞれローカルにレポジトリを持ち、そのローカルマシンにすべてのソースコードを複製することが可能であるため、多数の開発者が同一のソースコードについて同時に作業できるよう(同時並行で同一コードへの変更が可能)になっている。
- Git には、コード変更を行うか否かを決定する前に変更されたコードを一時的に保存するステージング(staging)システムがあり、組織のアプリケーションプロジェクト管理者は、同システムを用いることで、変更されたコードのビルドを行う前に、これらの変更点について検討し、一部の変更のみを承認し、他の変更点については保留するといった決定を行うことが可能である。

組織の中には、採用する開発プロセスに応じて、Git を異なる方法で導入している組織もある。たとえば、通常、Git を導入している組織では、どの開発者がどのソースコードについてどのような作業が進行中であるか、に関して完全な透明性が確保されている。

しかしオンラインおよびモバイル決済プラットフォームを開発

する米 Braintree Payments では、進行中の作業は他の開発者には見えないようにする形で開発者が変更コードをレポジトリに反映させる「プライベート・コミット(private commit)」を活用している。この理由として、頻繁にビルドを行っている Braintree では、作業中のすべての変更コードを公表しても管理者の手に負えず、特にコード変更が完全に完了していなかった場合や導入することが採択されなかったものまで全社的に開示する必要はないと考えているからである。同社では、Git システムにおいて、統合の準備が整ったとされる変更コードのみが公表されている[15]。

Git を活用した最新のプラクティスの一つに、ソースコードだけでなく、テキスト、ビデオファイル、画像等のデジタル資産の管理が挙げられる。

米 Center for Digital Information [16]の創設者、Jeff Stanger 氏[17]は、複数のプロジェクトで、プロジェクトのソースコードや画像ファイルなどを保存しておくためのホスティングサービス、Github に実際にエレメント文書を保存して Git を活用した経験を持つが、Git のバージョン管理、追跡、変更管理手法により、特定のアプリケーションまたはコードに関連したコンテンツモジュールの開発を効率的に管理することが可能であるとコメントしている。これらのプロジェクトには、多くの場合、顧客組織の関係者や外部のシステムインテグレータも参加し、エレメントの作成が行われており、こうした複数の機関が関係するプロジェクトを管理する上で、Git は特に有用であるとの見解を示している 。

3.3.2 バイナリ管理

単純なアプリケーション開発プロジェクトでは、ソースコードのレポジトリは、レビュー、承認、デリバリ向けに完全にコンパイルされたコードを格納する。一方、大規模かつ複雑なアプリケーション開発プロジェクトでは、アプリケーションに含まれるソースコードの行数が、ソースコードを管理する上で一つの大きな課題となっている。変更コードの複数のコンパイルバージョンが

---

15 https://www.braintreepayments.com/braintrust/our-git-workflow

16 公共政策機関を対象として、デジタルプレゼンテーション向けにレポートやウェブコンテンツの再設計を支援する米コンサルティング企業。
http://digitalinfo.org/

17 http://digitalinfo.org/director-jeff-stanger/

急速に増大し、従来のソースコード管理システムではこれに対応できない。

そこで注目される代替手段が「バイナリ管理」と呼ばれるものである。同手法では、レポジトリには生のマシンコード(バイナリ)が保存される仕組みとなっており、一般的に、異なる開発作業または変更作業は同一のバイナリコードをもとに行われる。バイナリレベルでソースコードを管理することは、追跡が必要なコードエレメントがかなり少なくて済むということを意味する。

バイナリ管理システムには、バイナリコードは関連するソースコードと関連付けて保存され、開発者は、レポジトリから特定の(ソースコードではなく)バイナリコードを複製し、変更を加えた後、コンパイル時に当該コードがどのような機能を果たすかについて定めたメタデータとともに、変更されたバイナリをレポジトリに登録する。また、新たなビルドに対するアプリケーションの準備が整っている場合に限り、バイナリコードはコンパイルされ、アプリケーションに統合される。

eBay の開発組織では、数千に上るソースエレメントから構成される単一のアプリケーションに関連したコードレポジトリをバイナリ形式で構成することで、従来のソースコード管理手法と比較して、ダウンロードおよびコンパイルが必要なコードエレメントの数をかなり減らすことに成功しており、リリース管理システムによる変更コードの追跡や、開発サイクルの終盤におけるコードエレメントと新たな変更箇所を一致させるプロセスを容易にしている[18]。

3.4 インフラの自動管理

アジャイル開発と DevOps、Continuous Delivery(CD)は、クラウドコンピューティングへの移行にも関係する場合が多い[19]。この理由として、クラウドコンピューティングプラットフォームは以下のような特徴を持ち、CI および CD へのアプリケーション開発手法への移行をサポートする機能を提供することが挙げられる。

---

18 http://www.ebaytechblog.com/2011/12/15/rapid-development-setup-in-large-environments/

19 CollabNet のホワイトペーパー(”Reinforcing Agile Software Development in the Cloud”)参照。 http://visit.collab.net/wp-agileincloud.html

- 分散型・高可用性インフラ
  定評のあるクラウドプロバイダは、ブロードバンドインターネット接続を通じてこうしたインフラへのアクセスを提供し、高いシステムアップタイム(99.9 %またはそれ以上)を保証する。これにより、企業の IT 部門は、さまざまな場所に開発チームを置くことが可能である
- 自動化されたソフトウェア配信環境
  数千人のユーザが利用するアプリケーションでは、クラウドプラットフォームは短期間でのソフトウェア配信を自動化するために非常に効率的な手段を提供する。これは、新たなプログラム(ビルド)をほぼ即座にリリースすることを目指す CD の目的に沿うものである
- 自動障害検出およびソフトウェアリカバリ
  クラウド環境では、自動テストスイート(test suite)を用いて、あらゆるソフトウェアの障害をユーザに配布する前に検知することが可能である。アプリケーション開発者は、クラウドインフラを活用することで、変更を加える前の安定した以前のバージョンのビルドに戻すことも容易となっている
- 適応性
  クラウドプラットフォームでは、パソコン、タブレット端末、スマートフォンといった複数のエンドユーザ端末に対応したアプリケーションの実装が可能であるため、開発者は、より柔軟に特定の端末のアプリケーションに対する開発、実装方法を選択できる

3.4.1 クラウドとアプリケーションライフサイクル管理(ALM)の自動化

クラウドソリューションを活用することは、IT 運用面で、完全に自動化されたアプリケーションライフサイクル管理(Application Lifecycle Management：ALM)インフラへの移行に近づけることが可能である。

DevOps 手法を採用したことのある開発グループは、少なくとも自動ビルド、テストツールを導入している可能性が高い。本番環境への自動リリースは、組織内のサーバよりもクラウドインフラを利用した方がより迅速に実現できることから、DevOps 機能を提供する新たなサービスを立ち上げているクラウドプロバイ

ダもある。たとえば、Amazon EC2 のクラウドプラットフォームは、Amazon Web Services(AWS)の提供する DevOps ツールキットと組み合わせて利用することも可能であり、CD システムを単純かつ迅速に構築するための一つの手段である[20]。

ALM ソリューションプロバイダは、提供ツールを DevOps およびクラウドコンピューティングに対応させることで、アプリケーション開発の自動化とアプリケーションの実装、計画を一体化させた機能を提供できる。アプリケーションの実装は IT 運用者の責任と見なされ、アプリケーションの「ライフサイクル」の外に位置付けられていたことから、従来型 ALM ソリューションはアプリケーションの構築、テスト機能に焦点が置かれている。しかし近年、ALM ベンダは、プロジェクト管理、アプリケーションのアーキテクチャ設計、アプリケーションの実装、およびキャパシティ計画機能を統合したツールを新たに提供するようになっている[21]。

多くの組織が、完全に自動化されたアプリケーション管理インフラを構築することは困難としており、アプリケーション開発および実装プロセスは、依然として手作業によるものとなっている。

具体的には、アプリケーションに不具合が検知されれば、自動アラートによりシステム管理者および開発者に通知されることになっているが、ほとんどの企業では、アプリケーションの復元(restore)プロセスは手作業で行う必要があり、自動化されていないほか、データベース、ネットワーク、サードパーティソフトウェアコンポーネント等の主要インフラコンポーネントの自動監視機能は、まだ一般に普及していない。

また、アプリケーション開発、リリース用に仮想化プラットフォームを導入している企業でさえ、インフラ運用者と開発者の間のコミュニケーションは人間同士の人的な接触によるもので、自動メッセージシステムは利用されていない。IT インフラの自動管理は CD および CI を実現する上で絶対不可欠な機能ではない

---

20 http://ovum.com/2013/04/02/businesses-adopting-devops-often-stop-short-of-continuous-delivery/

21 The Forrester Wave™: Application Life-Cycle Management, Q4 2012, published by Forrester Research on 23 October 2012. http://www.forrester.com/The+Forrester+Wave+Application+LifeCycle+Management+Q4+2012/fulltext/-/E-RES60080

が、Amazon(EC2)や Microsoft(Azure)を含むクラウドプロバイダは、こうしたトレンドに乗じる形で、アプリケーション開発に必要なインフラ管理機能も提供するようになっている。

3.4.2 クラウドと IT インフラの構築を自動化する Infrastructure as Code

クラウドソリューションの発展、普及により大量のマシンを構築、運用することや、物理インフラから切り離してソフトウェアおよびデータを管理することが可能となってきている。また開発したコードを即座に実装し、ユーザからのフィードバックを迅速に開発へ反映させる DevOps を実現する上で、こうしたインフラの運用管理を自動化するための手法「コードとしてのインフラストラクチャ Infrastructure as Code(3.2.1 参照)」が注目されるようになっている。

Infrastructure as Code は、IT インフラ(メモリ、ディスク、ネットワーク等のコンピュータ資源や OS、ミドルウェアを含む)のコンフィギュレーションといったインフラの構成管理をスクリプト化(コードに定義)して自動的に行う方法で、同スクリプトはソフトウェアのプログラムコードのようにバージョンを管理できることから必要に応じて変更を加えることが可能であり、加えられた変更は自動的にシステム構成に反映される仕組みとなっている。

クラウド環境で継続的インテグレーション(CI)プラクティスを実践する組織では、IT 運用者が扱わなければならないサーバインスタンスは非常に多い。従来の開発手法では、ソフトウェアのアップデートは数カ月単位で行われていたが、CI プラクティスを実践する組織では、毎日複数のビルドが作成され、そのテストを実行する必要があり、これを実現するためのインフラ構成環境をより迅速に構築することが運用者には求められる。

また、ビルドの中にはコンフィギュレーションの変更を伴う場合も多いが、サーバの管理コンソールからこうした変更をそれぞれ手作業で行っていたのでは多大な時間がかかり、手動設定によるシステムエラーの発生する可能性も高くなることから、開発サイクル(cycle time)は長期化し、定期的かつ迅速なソフトウェア機能のリリースを目指す開発チームの開発プロセスを妨げることになる。

アジャイル開発環境において、IT 運用者は、パッチの適用やコ

ンフィギュレーションの変更に必要な複数の環境を迅速に提供できるよう維持しなければならない。そして、パッチの適用またはコンフィギュレーションの変更にかかわらず、IT 運用者は、新たなバージョンの環境を設定して、それぞれ変更を行う必要がある。こうしたバージョン環境を効率的に管理する唯一の方法は、新たなバージョン環境の構築をスクリプト化して実行することである[22]。

Infrastructure as Code では、インフラ構成を一元的に管理するために定められるスクリプトに加えられた変更が、システムに自動的に反映される仕組みとなっている。そのため、継続的デリバリ(CD)ソリューションに特化したソリューションを提供する米 Stelligent の CTO を務め、DevOps の専門家である Paul Duvall 氏は、同手法では、同一のコンフィギュレーションを単一のノードにも数千のノードにも適用することが可能であり、環境を手作業で構成する場合と比較して、かなりエラーの発生しにくい方法で環境を複製できると説明する。

また、組織の開発チームは Infrastructure as Code を採用することで、各ノードごとに手作業でインフラ構成を設定する必要がなくなり、インフラ構成管理のためのスクリプトのバージョンを管理することによって、ソフトウェアのプログラムコードのようにインフラのスケールアップ(ダウン)も容易に行えるようになる。

Duvall 氏は、Infrastructure as Code によりインフラ構成を自動化することで、(1) 権限を持つ開発チームのメンバがアプリケーションの自動テストやバージョン管理等の優れた開発プラクティスを継続的にインフラに適用する環境を維持しながら、(2) 運用者はインフラの構成管理をスクリプトにより自動で実施することで、CI プラクティスの課題である煩雑なコンフィギュレーション変更の負担を軽減することができる、とした上で、このような手法は、アジャイル開発における開発者および運用者双方の俊敏性(agility)を高めることができるとしている[23]。

Infrastructure as Code 手法を適用するためには、Chef や Puppet といったオープンソースのインフラ構成管理自動化ツールを利用することが可能である。従来、Google や Amazon、Facebook といった大手ウェブサービス企業は、各社の保有する

[22] http://sdarchitect.wordpress.com/2012/12/13/infrastructure-as-code/
[23] http://www.ibm.com/developerworks/library/a-devops2/

非常に多数のサーバにおけるソフトウェアの実装およびコンフィギュレーションを自動化するツールを独自に開発していた。

しかし、2013 年 2 月、Facebook が同社のインフラ自動管理ツールに Chef(商用版「Private Chef[24]」)を採用したことが明らかになり[25]、こうしたツールのスケーラビリティの高さが業界でも話題となっている。15 万台以上という世界でも最大規模のサーバを保有、運用する Facebook では、世界の 10 億人以上のユーザが毎日サイトにアップする写真、ビデオ、メッセージ、ニュースフィードといった絶えることのない流入データをサポートするソフトウェアのコードをアップデートする作業を 1 時間に 2 度行う必要があるともいわれている。同社は、こうした膨大なサーバのソフトウェアコードのアップデート作業を迅速に行うために欠かせないサーバのコンフィギュレーションを自動化するツールとして、既存の cfengine2 ツールから新たに Chef を採用したことを発表した[26]。

2009 年頃に登場した Chef は、すべてが Ruby スクリプトとして定義され、Ruby 開発者の間で特に利用し易いツールとして知られているが、Chef の開発者で同ツールを販売する企業 Opscode の創設者の一人である Adam Jacobs 氏は、Facebook の運用する大規模なサーバ要件に対応するために Facebook と共同で新たな商用ツールの設計を行ったとしている[27]。

また、Facebook でプロダクションエンジニアを務める Phil Dibowitz 氏は、2013 年 4 月末に行われた Opscode のユーザ向けカンファレンス「#ChefConf 2013[28]」において、Chef を採用した理由について以下の要素を挙げている[29]。

---

24 http://www.opscode.com/private-chef/

25 http://www.opscode.com/press-releases/facebook-likes-opscode-and-private-chef/

26 http://www.opscode.com/blog/chefconf-talks/chefconf-2013-scaling-systems-configuration-at-facebook-the-paradigms-design-and-software-behind-managing-massive-numbers-of-systems-with-open-source-and-small-teams-phil-dibowitz/

27 http://www.wired.com/wiredenterprise/2013/02/facebook-chef/

28 http://chefconf.opscode.com/

29 http://developer.rackspace.com/blog/thoughts-from-chefconf-day-1.html

- 分散型システム
- 決定性(各動作において思い通りにシステムコンフィギュレーションを行える)
- 冪等性(べきとうせい：idempotent)(あるべき状態を把握した上でそれに必要な変更のみを行える)
- 拡張性(内部システムと結合できる)
- 柔軟性(決められたワークフローが存在しない)

Dibowitz 氏は、Facebook では、特定のソフトウェアとそれに付属するソフトウェア、設定をまとめた同 Chef ツールの Cookbook に各開発者が変更を加えられるようにすることで、1 万 7,000 台以上のサーバをわずか 4 人のチームで管理することに成功していると述べている。

なお、Puppet の開発元であり、インフラ構成管理自動化ツール分野で Opscode と競合する Puppet Labs の 創設者である Luke Kanies 氏は、「Opscode がこれほどのスケールに対応できるツールを提供するようになったことは感心する」と述べる一方、Puppet はすでに 3、4 年前から同スケールに対応できるようになっているとしている。同氏は、Puppet は、5 万台のサーバを所有する米オンラインゲーム開発企業の Zynga や、30 万台のサーバを運用することが見込まれているスイスにある欧州原子核物理研究所(CERN)で活用されているとし、こうした商用インフラ自動化ツールがデータセンタの継続的な改革につながっているとの考えを示している[30]。

[30] http://www.wired.com/wiredenterprise/2013/02/facebook-chef/

第4章
スタートアップ企業にみるアジャイル開発事例

4　スタートアップ企業にみるアジャイル開発事例

4.1　Etsy <www.etsy.com>

創設年	2005 年
本拠地	ニューヨーク州ニューヨーク
収益	推定 1,170 万ドル(2012 年)[1]
従業員数	400 名以上(2012 年)

4.1.1　企業概要

2005 年に立ち上げられた Etsy は、世界のアーティストおよびクリエーターによるハンドメイド品やビンテージ商品を販売する e コマースサイトで、これらの出品者[2]から、一商品の出品あたり 20 セントのリスティング料金と売上げの 3.5%を手数料として徴収している。2013 年 3 月時点で、Etsy に出店している販売者数は 80 万人以上で、販売商品数は約 1,800 万点、同サイトのユーザ登録者数は 2,200 万人に上っており、同サイトにおける年間商品総売上高は、図 4-1 に示すように、近年急速に増大している。

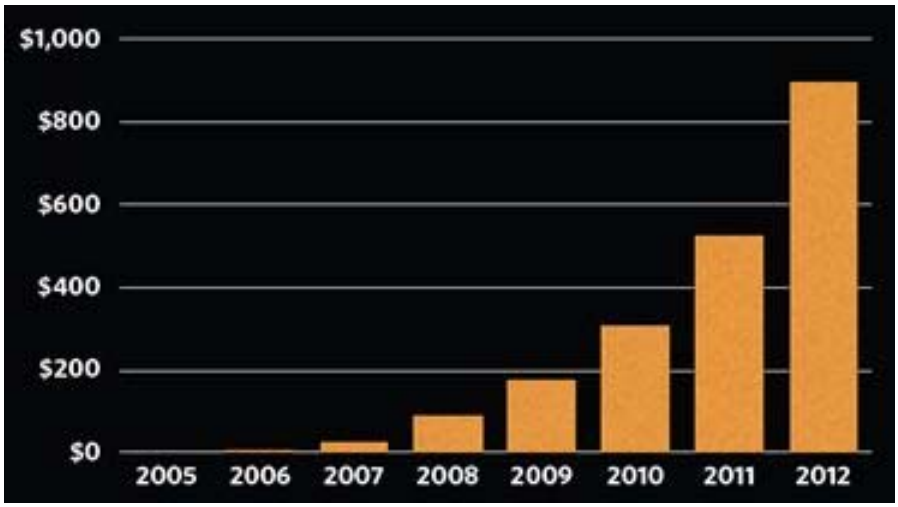


図 4-1 Etsy の年間商品総売上高の推移(出典 Etsy)

[1] Dun and Bradstreet's Global Duns Market Identifier
[2] Etsy は、再販業者による商品販売を禁じている。

4.1.2 アジャイル開発の実施状況に関する概要

(1) 急成長に伴う IT インフラの問題

このように、急成長を遂げている Etsy であるが、サイト立ち上げから 3 年後の 2008 年までに、同社の IT インフラとアーキテクチャではさまざまなニーズに対応できなくなったことで、その IT 開発、運用体制を見直すことが必要になった[3]。Etsy のソフトウェアエンジニアである Ross Snyder 氏は同社の当時の状況について「2008 年後半、Etsy はまだスタートアップであったが、サイト人気が急激に高まる中、同社の IT 開発、運用方法は事業運営が遅滞している老舗企業のようであり、機能の実装プロセスが追い付かない状況になりつつあった」と述べている[4]。

当時、Etsy の社内で、IT 開発者と IT 運用者との間のコミュニケーションはほとんどなく、開発者が書いたコードが、運用者による実装フェーズで多数の問題を引き起こすといった状況にあり、同社は、新たなコードの実装とウェブサイトのメンテナンスのために、長時間にわたるサイト休止を余儀なくされていた[5]。

同社の IT 開発者と IT 運用者との間のコミュニケーション不足は、同社の IT アーキテクチャにも起因しており、当時、フロントエンドシステムとデータベースの間に設置された自社開発の「Sprouter」と呼ばれるミドルウェアシステムに開発者は開発コードを提出するだけで、それが実装のために IT 運用者側に送られる仕組みであり、IT 開発者は、IT 運用者のシステム動作環境(データベース)を気にせず、コード開発のみに専念できるようなシステム設計となっていた。

そのため、運用インフラへの理解のないままコード開発を行う同社の開発者と、システムインフラの安定した稼働を望む IT 運

---

[3] Etsy, “Changing Etsy’s Architectural Foundation with Continuous Deployment,” September 28, 2012,
http://www.youtube.com/watch?v=nNOxJK6Urtg;
http://www.slideshare.net/mdg149/changing-etsys-architectural-foundations-with-continuous-deployment

[4] Continuous Deployment at Etsy: A Tale of Two Approaches,” Slide 20.

[5] Etsy, “Continuously Deploying Culture: Scaling Culture at Etsy,” October 4, 2012,
http://www.slideshare.net/mcdonnps/continuously-deploying-culture-scaling-culture-at-etsy-14588485;
http://itrevolution.com/one-of-the-best-devops-talks-on-it-transformation-continuously-deploying-culture-by-rembetsy-and-mcdonnell-velocity-london-2012/

用者の間では、新機能の実装をめぐる対立が生じていた。また、運用インフラに変更を加える際には、必然的に Sprouter にも変更を加える必要があり、急成長を遂げている Etsy にとって、こうした二重のシステム変更にかかる時間や費用が重圧となり、同ミドルウェアシステムがボトルネックとなりつつあった。

(2) アジャイル開発の採用

Etsy は 2009 年春、同社の直面するこうした技術的課題に対応し、新機能の開発環境における IT 開発者と IT 運用者との間の障害を取り除くため、IT インフラを刷新することを決定した。具体的に同社は、まず、IT インフラを単純化し、IT 開発者と運用者が各々のシステム理解を高めるため、システムで用いられているプログラム言語を統一し、フロントエンドシステムには PHP を、データベースには MySQL をそれぞれ標準として用いることを決定している。

Etsy では、PHP および MySQL を基盤とする新システムに IT インフラを移行させる期間、ウェブサイトを休止させる事態をどうしても避けたかったため、既存の IT インフラを、新システムに小規模かつ段階的に移行させることが可能なアジャイル開発を採用することを決定した。新プログラミング言語への移行の一環で、Etsy は、販売者が商品の写真を保存するために用いる既存の写真ストレージシステムを Python から PHP に、同社のウェブサイトのバックエンド基盤として機能するデータベースを PostgreSQL から MySQL に移行する必要があった。

同社は、この 2 つの移行作業をそれぞれ小規模の開発、実装、テストプロセスに分割し、一連の写真またはデータベースの表(テーブル)ごとにこれらのプロセスを繰り返すという方法で進めることを決定した。そして、新たな実装で欠陥コードが発見された場合に備え、同社は旧システムを再実装し、コードの修正と再実装を迅速に行える体制を整えている。

Etsy の主要エンジニアである Matt Graham 氏は、「小規模単位で迅速に実装可能なシステム変更を行える(アジャイル開発)手法を用いずに、こうした大規模な同社の主要システムの移行プロセスを実施しようとは考えなかった」と述べている[6]。

---

[6] Ibid.

Etsy のアジャイル開発は、既存システムから新システムへの移行、またはウェブサイトの再設計といった大掛かりなタスクを複数の小規模プロジェクト(Sprint)に分割する手法をとっており、同社は実装したい具体的なシステム新機能に関する 60 日計画を策定し、それを 2 週間ごとに完了できる小規模な開発計画に分割した[7]。開発のチーム体制は、単一のシステム機能開発プロジェクトごとに、3～7 名のエンジニア、設計者、商品管理者で構成されるチームが協力して作業にあたる仕組みであり、各チームはシステム機能の仕様を平易に保ちながら、新機能の最適な開発方法を自由に決定できるようになっている。

アジャイル開発によって実装する新機能の規模を軽減することで、問題のあるコードの分析、特定と、修正パッチの導入もより迅速に行えるようになるなど、Etsy はコードに欠陥があった場合の修復時間を短縮することに成功している。同社は、実装前に欠陥コードをすべて取り除くことは不可能であるとの考えから、そのために多大な投資を行うのではなく、アジャイル開発において、欠陥コードが特定された場合の平均復旧時間(mean time to recover：MTTR)を短縮することが得策であると考えた[8]。

また Etsy では、開発者は開発コードを実装前にテストできる一方、同コードによる諸問題を見極めるためには、システムが製品として実際に稼動している本番環境(production environment)でテストすることが最善の方法であると考えている。

Etsy は、開発された新コードの利用を当初は社内の従業員に制限し、新コードがウェブサイト機能に支障をきたさないかを確かめるためのテストを行い、新コードが正常に動作することを確認した上で、新コードによる変更が加えられたウェブサイトを利用できるユーザ(顧客)数を徐々に拡大する方法をとっている。新コードによる変更が加えられたサイトをすべてのユーザが利用できるようになるまでの期間は、新コードによるサイト機能の変更の度合いやテスト回数の必要性に応じて、1 日～6 週間と差があ

---

[7] Etsy, “How Does Etsy Manage Development and Operations,” April 2, 2011, http://codeascraft.com/2011/02/04/how-does-etsy-manage-development-and-operations/

[8] DZone, “Interview with John Allspaw, Senior Vice President of Technical Operations for Etsy,” September 28, 2012, http://java.dzone.com/articles/john-allspaw-discusses-devops; “Changing Etsy’s Architectural Foundation with Continuous Deployment,” Slides 15 – 19.

る。こうした本番環境においてテストを実施し、新コードの動作を見極めながらユーザ数を段階的に拡大する方法により、同社は、欠陥コードにより、サイトパフォーマンスが損なわれないようにしながら、アジャイル開発を行うことが可能となっている。

4.1.3 アジャイル開発の成功要因

Etsy におけるアジャイル開発の成功要因の一つに、IT 開発者と IT 運用者間とのコミュニケーションおよび協力を促す有効な「DevOps」文化を構築したことが挙げられる。

Etsy は、IT 開発部門と IT 運用部門の従業員の間で「信用(trust)」、「透明性(transparency)」、「コミュニケーション(communication)」、「協調(coordination)」を必要とする協力的な開発環境を築くことに成功しており[9]、欠陥コードにより生じるリスクを軽減しながら徐々に新機能をウェブサイトに追加することが可能となっている。また、同社の IT 開発者は、ウェブサイトに新たなコードを実装する時期について柔軟に決定し、開発コードの実装プロセスを自動化、単純化する「Deployinator」と呼ばれるオープンソースベースのツールを用いて、各開発者は自主的に開発コードを実装できるようになっている。

同社の IT 運用者は、かつて、IT 開発者の開発したコードを実装する際、同コードの問題がウェブサイトに不具合をもたらすことのないよう、その実装を注意深く監視する「門番」のような役割を担っており[10]、サイトの安定稼働を保証しなければならない IT 運用者は、IT 開発者とは相対する立場にあった。

しかし Etsy は、開発コードの質は本番環境で初めて確認できるとの考えの下[11]、上述のように、テストを実施し、新コードの動作を見極めながらユーザ数を段階的に拡大するアジャイル開発方法を採用したことで、IT 運用者は、開発者がより質の高いコードを開発できるよう支援する立場となり、その役割および IT 開発者との関係は大きく変化した。具体的には IT 運用者は、開

---

[9] How Does Etsy Manage Development and Operations."
[10] Interview with John Allspaw, Senior Vice President of Technical Operations for Etsy."
[11] Etsy, "Continuous Delivery: The Dirty Details," January 29, 2013, http://channel9.msdn.com/Events/ALM-Summit/ALM-Summit-3/Continuous-Deployment-The-Dirty-Details; http://www.slideshare.net/mikebrittain/continuous-delivery-the-dirty-details (Slides 89 -- 91)

発者のアジャイル開発プロセスを支援するため、ウェブサイトのパフォーマンスを測定できる多数の測定値監視システムを開発し、開発コードが本番環境に与える影響や、テスト時に監視すべき基準値について IT 開発者の理解を深めるために支援を行っている。そして、こうした相互のコミュニケーションを通じて、IT 開発者はウェブサイトの本番環境について、IT 運用者は新機能がどのように動作し、コードがどのように構築されたかについて、それぞれ理解を深めることが可能となった[12]。

4.1.4 アジャイル開発の課題

アジャイル開発において Etsy が直面している課題の一つに、DevOps 体制を同社のクレジットカード決済処理システムに反映させることへの限界が挙げられる。

Etsy の IT 開発者は、アジャイル開発において、実装可能と判断された段階で、開発コードを運用インフラに直接実装できるようになっている。しかし、同社が自社開発、運用しているクレジットカード決済処理システムはクレジットカード業界のセキュリティ対策基準(Payment Card Industry Data Security Standard：PCI DSS[13])に準拠する必要があることから、開発者は新たな開発コードをサイトの決済処理システムに実装する際、社の了承を得た上で、実装は IT 運用者が担うことになっている。そのため、決済処理システムについては、Etsy は他のシステムと同等の DevOps および迅速な機能開発、実装プロセスを実現することができない状況にある[14]。

---

12 Interview with John Allspaw, Senior Vice President of Technical Operations for Etsy."

13 VISA、MasterCard、Discover Financial Services、American Express、JCB International の国際ペイメントブランド 5 社が共同で策定したクレジットカード情報・取引情報を保護するためのセキュリティ標準で、公衆ネットワーク上でカード会員データを送信する場合に暗号化するといった規定が含まれる。（https://www.pcisecuritystandards.org/security_standards/)

14 "Changing Etsy's Architectural Foundation with Continuous Deployment;" DZone, "Continuous Deployment and PCI-DSS at Etsy," July 15, 2012, http://java.dzone.com/articles/continuous-deployment-and-pci

4.1.5　アジャイル開発の実施結果

(1) サイトパフォーマンスへの影響を最小限に抑え、新機能の実装および IT インフラの大規模刷新に成功

Etsy は、小規模な機能開発を継続して行ってシステムに変更を加えるアジャイル開発手法を採用したことで、より迅速に新機能を実装することができるようになっている。

図 4-2 は、Etsy における新機能の実装頻度を示したもので、2010 年はじめ時点においては、同社の 1 カ月当たりの平均実装頻度は数えられるくらいの頻度にとどまっていたが、2012 年に同頻度は 1 日当たりおよそ 25 回にまで増加していることが読み取れる。なお、図中の Web の線(黄)はフロントエンドシステムにおける実装頻度を示しており、config の線(赤)はデータベースにおける実装頻度を示している[15]。なお、図中の赤枠は、米国の感謝祭からクリスマスにかけてのホリデー・ショッピング・シーズン期間を示したもので、同期間には Etsy のサイトアクセスおよびトラフィックが増大することから、サイトの維持、運営が通常より重視される時期でもある。同期間は、ウェブサイトに問題が発生することを極力避けるため、同社はサイトへの主要な機能変更を行わず、微細な機能変更にとどめている。

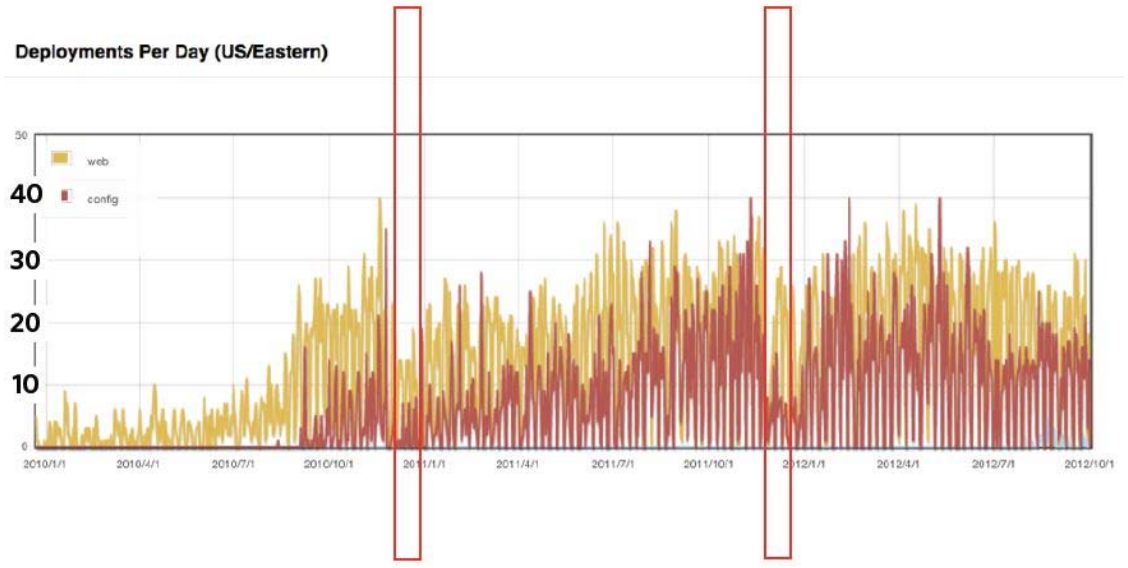


図 4-2　Etsy における新機能の実装頻度(出典 Etsy)

15　“Continuous Delivery: The Dirty Details,” Slide 116.

また Etsy は、ウェブサイトのパフォーマンスを損なわずに新機能を迅速に実装できるアジャイル開発により、同社の事業を危機にさらすことなく、大規模な IT インフラ刷新計画を推進することに成功している[16]。

(2) 「継続的な実験(Continuous Experimentation)」手法の採用により、ユーザ需要の高い新機能を導入することに成功

また Etsy は、アジャイル開発により、顧客が新サービスおよび新たなサイト設計に満足しているかについて、より正確に把握し、顧客にとって利用価値の高いウェブサイトの設計を強化できるようになっている[17]。

同社はかつて、サイトに新サービスを導入する際、一部の顧客に新サービスを導入したサイトを利用してもらうことで、新サービス導入前のサイトと比較して、顧客のサイト活用方法にどのような変化があったかを分析していた。

しかし、同方法では、新たに導入されたサービスに顧客が好感触を示さなかった場合、同サービスに含まれるどの機能が最も活用されていないかについて具体的に特定することが困難なことが問題となっていた。

そこで Etsy は、新サービスにおいて顧客の好む機能を分析し、より明確に特定できるようにするため、アジャイル開発において、同社が「継続的な実験(Continuous Experimentation)」と称する新たな開発戦略を採用している[18]。同開発戦略は、大掛かりなサイトへの機能変更を行う際、複数の機能を一度に開発、実装するのではなく、各機能変更への顧客の反応をそれぞれ個別に把握できるよう、機能をより小規模な単位に分割し、段階的に開発、実装する方法である。

Etsy は 2012 年秋、潜在的な商品の買い手による商品販売状況に直接影響するウェブサイトの主要機能である検索機能の再設計プロジェクトに着手した際、新たな開発戦略を活用した。このプロジェクトにおいて同社は、それぞれ独立して開発可能な機能単位にサービス機能を分割し、各機能変更に対する顧客の反応を

---

16 “Changing Etsy’s Architectural Foundation with Continuous Deployment;” Slides 66 – 69.
17 Etsy, “Design for Continuous Experimentation,” December 2, 2012, http://mcfunley.com/design-for-continuous-experimentation
18 Ibid.

個別に測定できるようにしており、各機能に対する顧客の反応を分析した結果、同社は、一部の機能が顧客によりまったく利用されていないことを把握することに成功している。

たとえば、Etsy は当初、ウェブサイトの左端に、買い手が商品の検索結果を各カテゴリ別に分類したリンク機能を追加(図 4-3)したが、同機能はほとんど利用されなかった。そのため、同社はこの機能を削除し、代替案として検索バーに入力された文字に基づくキーワードを提案する AutoSuggest および自動カテゴリ分類機能を追加(図 4-4)、顧客による同機能の活用状況を分析した結果、顧客は検索バーに表示された検索商品の自動カテゴリ分類機能を多用する傾向にあることが判明した。

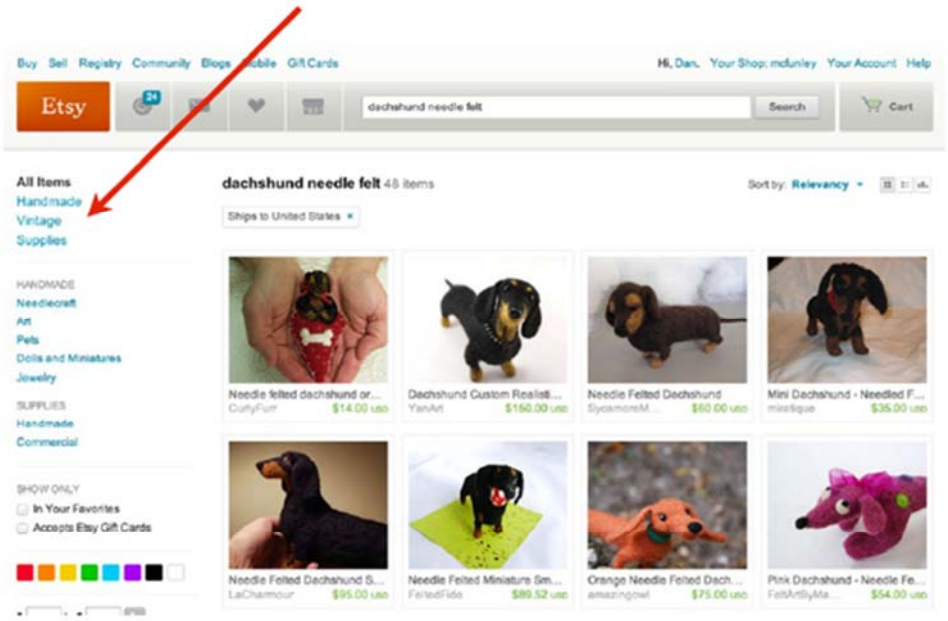


図 4-3 検索結果のカテゴリ別分類機能(出典 Etsy)

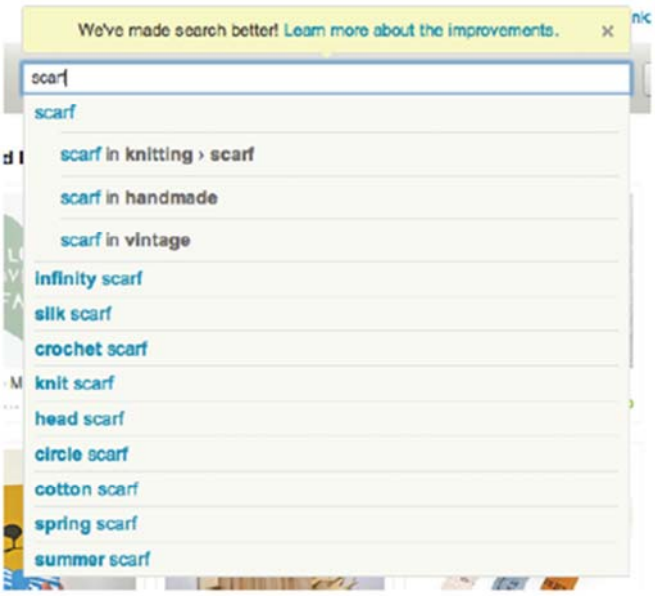


図 4-4　検索結果のカテゴリ別分類機能(変更後)(出典 Etsy)

また、検索商品の自動カテゴリ分類機能を検索バーに追加した Etsy のウェブサイトへのアクセスユーザ(買い手)によるビンテージ商品の売上が、同機能を持たないサイトを利用している顧客のそれと比較して 3.7%増加した。新機能の追加により、さらに明らかになったのは Etsy がウェブサイト上でハンドメイド品だけでなくビンテージ商品も扱っていることを多くの買い手が認識していなかったことである。

このように、継続的な実験に基づくアジャイル開発により、Etsy は、サイトに追加された新たな機能や設計において、具体的な変更点とユーザの反応をより詳細に把握できるようになっており、同社は継続的開発戦略を今後も採用する方針である。

4.2　Zynga<www.zynga.com>

創設年	2007 年
本拠地	カリフォルニア州サンフランシスコ
収益	12 億ドル(2012 年)
従業員数	2,850 名(2012 年)

4.2.1　企業概要

Zynga は、カリフォルニア州サンフランシスコに本社を置く

2007 年創設のソーシャルゲーム開発企業。同社は Facebook やモバイル端末、オンライン(Zynga.com)などのプラットフォームで動くゲームを開発しており、基本的なゲームを無償でユーザに提供し、高度かつ特別なゲーム機能について有償提供する「フリーミアム(freemium)」と呼ばれるビジネスモデルで急成長し、2011 年 7 月に IPO を申請し、同年 12 月に上場企業となっている。

4.2.2 アジャイル開発の実施状況に関する概要

Zynga は、新ゲームおよび既存ゲームの新機能のリリースまでにかかる開発期間を短縮するためにアジャイル開発を採用している。従来、ゲーム開発には数年を要することが一般的であったが、Zynga はアジャイル開発により、数週間で開発することを可能とした。Zynga の大ヒットソーシャルゲームの一つ、「FarmVille」の開発リーダーである Amitt Mahajan 氏は「とにかく早く何か作りたかった。当初、我々はゲームの構想から立ち上げまで 8 週間で行うことを想定していたが、開発プロセスは予想以上に早く進み、実際にかかった期間は 5 週間であった」と述べている[19]。

またアジャイル開発は、既存ゲームに新機能を次から次へと迅速に追加することを可能にしており、プレイヤを飽きさせず、長期にわたって惹きつけることにも成功している[20]。同社では、2～3 の開発チームが同時並行で相互に連携しながら既存ゲーム(1 タイトル)の新機能を開発しており、毎週、複数の機能をリリースするという開発サイクルを維持している[21]。

4.2.3 アジャイル開発の成功要因

Zynga によれば、同社のアジャイル開発の成功要因は、学際的な開発チームの構成と各チームメンバが相互に継続してコミュニケーションがとれる環境形成にあると考えている[22]。従来のゲ

---

[19] Escapist Magazine, “How FarmVille Was Written in Five Weeks,” March 8, 2010,
http://www.escapistmagazine.com/news/view/98990-How-FarmVille-Was-Written-In-Five-Weeks

[20] Todd Warren’s Blog – Facebook Developer Meet-Up/Zynga FarmVille,
http://web.archive.org/web/20091222075715/http://h177870wp.setupmyblog.com/?p=117

[21] Ibid.

[22] Gamasutra, “IDGA Leadership Forum: Sega Australia, Zynga Discuss the

ーム開発プロセスにおいては、ゲームデザイナがかなり詳細に仕様を策定し、その後、開発者が仕様を実現するために開発に従事するといった工程をとる一方、Zynga では、ゲームデザイナとゲームの品質保証を行うテスタが開発者と共同でゲーム開発を行っている。

同社は、ゲームデザイナが開発者と共同で仕様の策定にあたることで、デザイナは新機能に係る構想を早い段階で開発者と共有し、仕様の要求事項への改善点に関する開発者からの洞察を得られる。また開発者がデザイナのゲーム構想をより良く理解し、その最適な実現方法を効率的に決定できるため、特にゲームデザイナと開発者が共同で開発にあたることのメリットは大きいと考えている[23]。

4.2.4 アジャイル開発の実施結果

新ゲームおよび既存ゲームの新機能の迅速な開発を実現している Zynga は、Etsy と同様、継続的な実験に基づくアジャイル開発戦略を採用し、新ゲームおよび新機能へのユーザの反応を分析、調査している。

また同社は、特に新ゲームの開発に多大な投資を行う前に、同社が「ゲットー・テスティング(Ghetto Testing)」と称するプロセスを用いて、プレイヤが、そのゲーム構想に関心があるか否かの見極めを行っている。同プロセスは以下のようになっている[24]。

(1) 開発者は、Zynga の新ゲームに関する構想を、マーケティング担当者またはプロダクト・マネージャに伝える[25]
(2) 新ゲームについて広告し、同ゲームのプレイを希望するプレイヤに電子メールアドレスを送信するよう要請する
(3) 多くの機能が未完成であるが、ユーザがゲームの面白さを判断で

Agile Way," November 4, 2010,
http://www.gamasutra.com/view/news/31365/IGDA_Leadership_Forum_Sega_Australia_Zynga_Discuss_The_Agile_Way.php

[23] "How FarmVille Was Written in Five Weeks."

[24] Stanford University, "Zynga: Ghetto Testing and Minimum Viable Product," January 9, 2012,
http://ecorner.stanford.edu/authorMaterialInfo.html?mid=2313

[25] 初期構想段階で多大な時間及びリソースを割かないようにするため、ここでの新ゲーム構想は単純で説得力を持つ 5 ワード以下で表現する必要がある。

きる最低限の状態まで開発されたゲーム(「ゲットー版(ghetto version)」と称する)を開発する
(4) ゲットー版を Zynga の全ユーザ層[26]の 1～10%に提供する
(5) 各プレイヤがゲームをどのようにプレイしたかに関するデータを収集し、同社のテスト環境内のデータウェアハウスに保存する
(6) 収集データを分析し、ゲットー版から正規ゲームへの開発を進めるか、開発を打ち切るかについて決定する(具体的な判断基準の詳細は不明)

このように、大規模なユーザ層とアジャイル開発プロセスに基づく迅速な新ゲームおよび新機能の開発能力を組み合わせることで、Zynga は、新ゲームのプロトタイプを迅速に開発し、ゲットー・テスティングにおけるデータ分析結果をもとに、同社が開発予定の新ゲームへのプレイヤの関心度を早期に把握、新ゲームの開発にどの程度リソースを配分すべきかについて適切な判断を行えるようになっている 。通常、ゲーム開発には少なくとも 2 年以上を要する状況において、アジャイル開発を採用している Zynga の強みは、プレイヤの関心の高いゲーム(あるいは機能)を的確に把握し、プレイヤを飽きさせない機能を短期間のサイクルで提供し続けている点にあるといえる。

※(参考)事業低迷が継続、大型レイオフも実施予定
IPO を果たした当初、時価総額 70 億ドルを記録した Zynga は、競合企業からのデザイナ引き抜きなども含め 1 年間に 1,000 人以上を雇い入れ、新ゲーム開発に一層注力したが、モバイルゲーム市場における新ビジネスが不振だったり、多くのゲームの人気が極めて短期的な一過性なものに終わるなどの要因もあり事業低迷が続いている。2013 年 6 月には 500 人以上のレイオフを予定していることが発表され、株価も低迷し続けている。

4.3 Huddle<www.huddle.com>

創設年	2006 年
本拠地	ロンドン(英国)
収益	不明
従業員数	不明

[26] Zynga の月間アクティブユーザ数は 2 億 3,200 万、日間アクティブユーザ数は 6,000 万である。http://company.zynga.com/news/fact-sheet

4.3.1 企業概要

Huddle は、ドキュメントの共有と管理、およびプロジェクトの一元管理を行えるクラウドベースのビジネスコラボレーションサービスを開発、提供する企業で、同社は同サービスを月額15 ポンド(ユーザ 1 人当たり)で提供している 。同社のサービスは、Microsoft のビジネスコラボレーションソフトウェア、「SharePoint」と競合しているが、SharePoint はサーバ中心のソリューションであるのに対し、Huddle のサービスは、さまざまな拠点にいるユーザがより利用し易いアーキテクチャであるクラウドベースのサービスであることが競争上の強みの一つである[27]。

Huddle は、英国政府のクラウド展開戦略である「G-Cloud Framework [28]」を通じた政府調達用クラウドサービス、製品のうち、3 番目に販売数の高いサービスとなっており、英中央政府機関の 80%、および Fortune 500 にリストされる大企業の 80%を含む 10 万社以上の企業が同社のサービスを利用している[29]。

4.3.2 アジャイル開発の実施状況に関する概要

Huddle のコラボレーションサービスは、当初、サードパーティ開発者が開発を行い、その後社内で継続的に開発を進めるというプロセスを経ていた[30]。同社が Jonathan Howell 氏を同社の CTO に起用した 2008 年 3 月当時、Huddle には明確な開発プロセスが存在せず、アジャイル開発を強く支持する Howell 氏は、アジャイル開発手法を取り入れた開発プロセスを構築した。

Huddle の共同創設者で戦略部門のエグゼクティブ・バイスプレジデントを務める Andy McLaughlin 氏は「当時、Huddle はまだスタートアップであり、すでに確立した開発文化もなかったため、明確な開発プロセスを規定するのに良いタイミングであっ

---

[27] Computer Business Review, “There is a Lot Wrong With SharePoint: Q&A With Huddle,” August 13, 2012.
[28] HM Government – G Cloud Programme, http://gcloud.civilservice.gov.uk/
[29] Huddle, “Huddle Reveals Record Number of Government Deals,” January 17, 2013,
http://www.huddle.com/about/news/press-releases/huddle-government/
[30] ThoughtWorks Studios,” Huddle Case Study,”
http://www.thoughtworks-studios.com/customers/huddle

た」と述べている[31]。

Huddle は、同社のすべての開発プロジェクトにアジャイル開発を採用しているわけではない[32]。同社は、小規模かつ徐々に機能変更を行う必要のあるシステム開発プロジェクトにアジャイル開発を採用しており、顧客および販売担当者から得たフィードバックをもとに、サービスの問題箇所を分析、特定し、小規模な機能単位に分割してシステム変更を行う 2～3 のプロジェクトチームがアジャイル開発を担っている[33]。

これらのチームは、通常、アーキテクト(1 名)、開発者(2～3 名)、フロントエンドユーザインターフェイス(UI)開発者(1 名)、サービス品質を保証する専門家(1 名)、IT 運用者(1 名)から構成され[34]、2 週間の開発サイクル(Sprint)で機能変更を実施し、2～3 の Sprint サイクルを経て一つのシステム変更を完了させることが一般的である。

一方、Huddle は大規模なシステムプロジェクトについては、着手段階で仕様および要求事項をかなり明確に規定することが、開発プロセスをスムースに進めるために重要であると考えているため、アジャイル開発を採用していない[35]。

4.3.3 アジャイル開発の成功要因

Huddle は、アジャイル開発プロジェクトに携わる開発チームに、IT 運用者を含めることの重要性を指摘している[36]。IT 運用者は、新たなプログラムコードとその作成を行った開発者について把握し、当該コード実装後にサービスを適切に運用するため、必要に応じてコード修正を行えるよう備える必要があるからである。

---

31 Dev2Ops, “Huddle.com: Being Agile About Agile With Andy McLaughlin,” August 13, 2011, http://dev2ops.org/2011/08/huddle-com-being-agile-about-agile-with-andy-mcloughlin/

32 Ibid.

33 DZone, “Sales Driven Development,” February 27, 2012, http://agile.dzone.com/articles/sales-driven-development

34 Huddle.com: Being Agile About Agile With Andy McLaughlin.”

35 Ibid.

36 Ibid.

4.3.4 アジャイル開発の課題

Huddle は、アジャイル開発により、特定のシステム変更を完了するために必要な推定期間を特定し、各開発者にそれぞれが担当する開発の期限を順守する責任を負わせ、開発プロセスの透明性を確保し、定められた期間内に開発を完了させることに成功している[37]。

特定の機能の開発を完了するためにかかる労力、開発の複雑さ、付随するリスクなどを考慮して、その推定開発期間(ストーリーポイント(story point))は、アジャイル開発を成功に導くために求められる重要な要素の一つである。しかし Huddle の開発者は、ある機能を開発するのにどの程度のストーリーポイントが必要であるかの推定算出方法と、同ポイントを企業コストや価値、製品の開発ロードマップ全体を完了するのに必要な時間と関連付ける方法を習得することに苦心している[38]。

ストーリーポイントの推定算出能力を高め、ストーリーポイントと企業全体のコスト、価値、時間との関連性への理解をすべての従業員に促すための一つの方法として、Huddle では、「エクスペリエンス・ゲーム(Experience Game)」と呼ばれる社内イベントを開催している[39]。これは、同社の開発者だけでなく、販売担当者等、他の部門の従業員も加えて組織横断型のチームを複数構成し、ゲームのために選定したある特定の機能について、ストーリーポイントの推定算出を行うものである。同ゲームは、開発者以外の他部門の従業員が、開発者がどのようにストーリーポイントを算出しているか、また一つの機能の開発を完了させる上で開発者が直面する技術的課題などについて理解を深め、実際の現場で、新機能の開発、実装に必要な期間について開発者とコミュニケーションを図る際、より現実的な議論を行うことに役立っている[40]。

4.3.5 アジャイル開発の実施結果

アジャイル開発により、Huddle は、顧客からのバグ修正やシステム変更、新機能の追加といった要請により迅速に対応するこ

---

37 Ibid.
38 Huddle, “Developer Thoughts on Agile Process, Cost, and Value,” July 24, 2009, http://www.huddle.com/blog/agile-development-process/
39 Huddle.com: Being Agile About Agile With Andy McLaughlin.”
40 Ibid.

とが可能となっている[41]。

特に、新機能の開発要請に迅速に応えられるようになったことで、Huddle では、販売指向型開発(Sales-Driven Development)プロセスを採用し、サービス機能の向上や売上拡大を積極的に推進するようになっている[42]。具体的には、同社のサービス製品の販売が成功した際には、販売担当者が顧客の製品購入につながった機能を特定し、また販売につながらなかった場合には、製品購入につながる Huddle 製品への追加機能や変更機能をそれぞれ特定するようにしており、こうしたデータを集計した分析結果をプロダクト・マネージャおよび開発者に報告することで、Huddle に必要なサービス機能の変更や追加機能について効率的に検討、決定できるようにしている。

4.4 Halliburton：Landmark Software & Services

Landmark Software & Services(以下、Landmark と称する)[43]は、1919 年創設の米原油採掘関連サービス大手 Halliburton の一部門で、石油およびガス業界向けに、原油採掘施設の建設や運営、管理をサポートするための地震データ分析ソフトウェア製品を(顧客システムへの導入またはプライベートクラウド上でのホスティングサービス形態で)提供している。

4.4.1 Landmark Software & Services がアジャイル開発を採用した理由

Landmark は、顧客のニーズに沿ったソフトウェアを開発、提供できなければ、競合事業者にビジネスを奪われるとの認識を常に持っており、同社のソフトウェア開発ではアジャイル開発プロセスがより適していると判断し、2006 年からアジャイル開発を採用するようになっている。アジャイル開発を採用したことで、同社は、顧客のニーズに対応した高品質のソフトウェアを迅速に

---

41 Programmable Web, “Huddle API: Facebook Meets Sharepoint,” September 22, 2008, http://blog.programmableweb.com/2008/09/22/huddle-api-facebook-meets-sharepoint/

42 “Sales Driven Development.”

43 Landmark Software and Services は、1919 年創設の米原油採掘関連サービス大手 Halliburton の一部門で、石油およびガス業界向けに、原油採掘施設の建設や運営・管理をサポートするための地震データ分析ソフトウェア製品を（顧客システムへの導入又はプライベートクラウド上でのホスティングサービス形態で）提供している。https://www.landmarksoftware.com/

開発できるようになっている。

Landmark がアジャイル開発を採用した第一の理由は、要求の変更に応じて、より迅速にソフトウェアに修正を加えられることにあり、要求定義に多くの不確定要素を伴い、継続的な修正が必要なソフトウェア開発にはアジャイル開発が適していると考えたためである。第二の理由は、ソフトウェアの品質を向上させるためである。

Landmark では、かつてプロジェクト管理者、製品管理者、開発者、テスタなどからソフトウェア製品チームを構成し、開発者がソフトウェア開発を完全に完了するのを待ってから、欠陥テストの実施とプログラムコードの修正を行い、特定された欠陥がすべて取り除かれたことを確認する作業を進めていた。しかし、開発者の開発するソフトウェアには欠陥が多く、開発完了後から顧客への製品提供期限までの期間に、すべてのバグを修正するために十分な時間を確保できなかった。期限の延長が不可能な場合には、確認された欠陥を未修復のまま顧客に提供し、後日修正パッチまたは次のバージョンソフトウェアを提供することで対処していた。労働集約的なバグの特定とコード修正、迫られる納期に開発チームはストレスフルになり、開発者の離職率も高かった。

Landmark では、アジャイル開発の採用後は、1～3 週間ごとの短期間サイクルを通じてソフトウェア機能を分割して開発する手法をとっている。しかし、アジャイル開発の導入当初は、開発チームが従来のテストプロセス習慣をすぐには逸脱できず、期日通りに質の高いソフトウェアの開発を一貫して行うという目標を達成できずにいた(後述)。こうした事態を打開するため、Landmark は 2010 年以降、分割された各開発フェーズを通じてアプリケーションのテストプロセスを実施し、問題を迅速に修正するようにし、組織文化の変革や、一日ごとにテスト結果を表示することが可能な自動ツールを導入するなどし、状況の改善を図っている。

4.4.2 アジャイル開発の課題

アジャイル開発の導入に際し Landmark が直面した課題の一つに、期日までに迅速かつ一貫して質の高いソフトウェアを開発するため、既存のテストプロセス、ソフトウェア開発プロセスを改善する必要があったことが挙げられる。しかし、ソフトウェア

の開発が完全に完了してからテストを実施するという習慣が組織の開発チームの中で深く染みついており、同プロセスを変えることは容易ではなかった。

Landmark は、まず開発プロセス全体を通じてバグの特定を行うよう、テストプロセスを改めたが、開発チームはバグの特定はするものの、すべてのサイクルにおける機能開発が完了するまではコード修正しなかったため、テストの継続的な実施と修正を通じたプログラムの品質向上を図るというアジャイル開発のメリットを生かすことができないでいた。

また、Landmark にはかつて、販売促進用のデモ版ソフトウェアの開発を奨励するインセンティブプログラムが存在した。このソフトウェアは品質よりも最低限の動作を保証すれば良い程度のものだったため、本来のソフトウェア開発プロジェクトにおいても開発者は質の高いソフトウェアを開発しようというインセンティブが働かず、結果的にデモ版相当のソフトウェアを生み出す要因の一つとなっていた。

Landmark は開発チームに根付いているこのような文化的な問題を全面的に見直し、バグのほとんどない質の高いソフトウェアを開発した開発者に報酬を与えるといったように社内のインセンティブプログラムを改めた。同社で開発を担当する幹部は、Landmark がアジャイル開発の導入に際し直面した問題の大部分はこうした文化的な問題であり、開発、テストツール等の導入に関する技術的な問題は特になかったとの見解を示している。

Landmark の幹部は概してアジャイル開発の採用を支持しており、これは、ソフトウェア品質や契約期限を保証できなければ、顧客を競合事業者に奪われるリスクを認識しているためだと Little 氏は考えている。

一方、組織内でのみ利用するシステム開発を手がける Halliburton の IT 部門にはこうした競争概念がないため、同社の幹部は従来の開発手法を改革する必要性はないとみている。しかしながら、クラウドベースの IT サービスが成長を遂げる中、サードパーティクラウドサービスプロバイダがこうした IT 部門の競合事業者となる可能性はあり、その脅威を認識し、同部門がアジャイル開発の採用に動くこともあり得るのではないかとも考えている。

4.4.3 アジャイル開発の実施結果

アジャイル開発の採用と、その後のテストプロセスの改善により、Landmark はソフトウェアのバグ数を削減するとともに、顧客へのソフトウェア配布時に特定されているバグ数を 2009 年から 2010 年にかけて 97%削減し、ソフトウェアが顧客の手に渡る前に確認されている大部分のバグを修正することに成功している[44]。

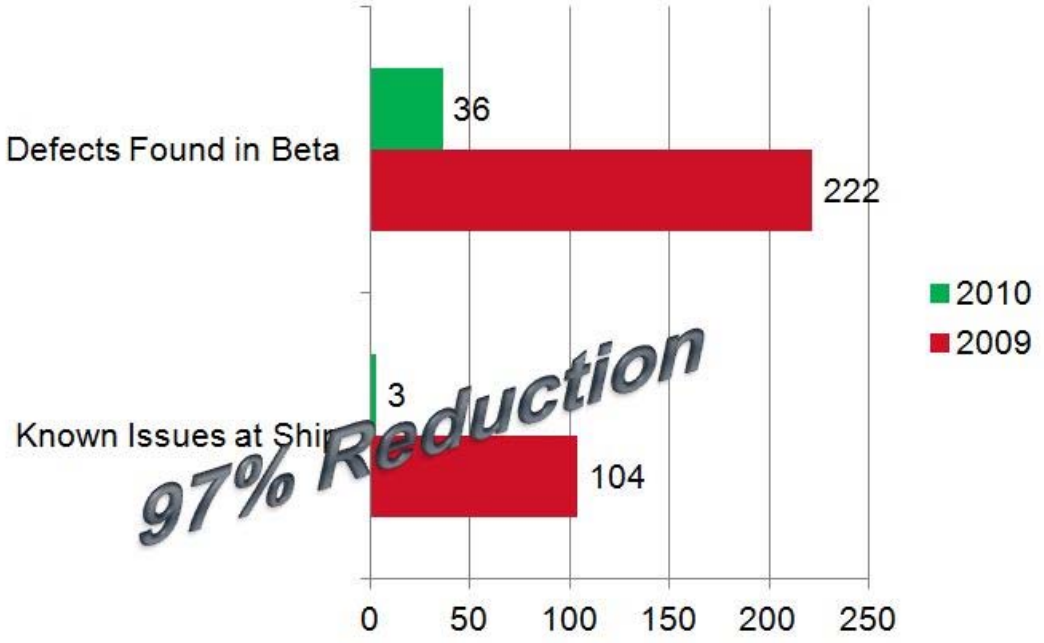


図 4-5　アジャイル開発導入およびテストプロセス改善後のソフトウェア品質改善状況(出典 Landmark Software and Services)

4.4.4 アジャイル開発がクラウドベースのサービス開発者にもたらすメリット

Landmark を含む多くのソフトウェア開発者が抱える問題の一つに、顧客がソフトウェアのバージョンアップをなかなか行わないことが挙げられる[45]。ソフトウェアを新バージョンに移行する

---

[44] Todd Little, “Leveraging Global Talent for Effective Agility,” 2012, http://www.toddlittleweb.com/Presentations/Gartner2012/GlobalAgilityGartner2012final.pptx

[45] Landmark では、顧客システムへの導入またはプライベートクラウド上でのホスティングサービス形態でソフトウェアを提供しているが（石油・ガス業界の顧客は IT 管理への考え方が概して保守的であることから)、顧客の大部分は、ホスティングサービスよりもシステム導入を好んでいる。

ためにはコストがかかり、特にそれが顧客システムに導入されている場合は負担がより大きいことから、顧客の中には、古いバージョンのソフトウェアを利用し続けるケースもある。クラウドサービスは、クラウド上でソフトウェアを活用するすべての顧客に対し、開発者が最新版のソフトウェアの活用を強制できることが一つの利点である。

アジャイル開発は、クラウドベースのサービスの利点を強化するものである。アジャイル開発は、開発プロセスを通じて絶えずテストを行い、高品質のソフトウェアを迅速に開発するプロセスを支援し、クラウドサービスは、同プロセスにより開発された新機能を含む最新版のソフトウェアを顧客が迅速に利用することを可能にするものである。アジャイル開発とクラウドベースのサービスを組み合わせることで、開発者は、顧客から得たフィードバックをもとに、顧客の望む新たな機能の追加や機能変更を迅速に行うことができる。

第5章

まとめ

5　まとめ

本章では、米国のアジャイル開発や DevOps の成功事例から得られる示唆を考察するとともに、日本のシステム開発における課題と、新しい手法や技術を導入する上で求められる組織や文化の転換について述べる。

5.1　米国成功事例からみる考察

(1) 顧客からのフィードバックプロセスの組み込み

Etsy の「継続的な実験（Continuous Experimentation）」
Zynga の「ゲットー・テスティング（Ghetto Testing）」
Huddle の「販売指向型開発プロセス（Sales-Driven Development）」

米国のスタートアップ企業では、必ず顧客の反応や利用状況を把握するプロセスを開発の中に組み込んでいる。いかに早く、正しく顧客からのフィードバックを得て、それを開発に生かせるかが成功の鍵と言える。

(2) 組織・文化の転換

米国の成功事例において、各企業は 5.1 (1)のようなさまざまな手法を取り入れていることで成功しているが、手法だけを真似してもうまくいかない。手法と同等に大事なことは「組織・文化の転換」である。

たとえば、Landmark はビジネスの危機感を感じた組織幹部の強い支持でアジャイル開発を導入したが、それまでデモ版ソフトウエアの開発に報酬を与えて推奨することが、品質低下の根本原因と捉え、バグの少ない高品質なソフトウエアの開発者に報酬を与える、といったインセンティブプログラムの改定を行って成功している。

「組織・文化の転換」は、具体的には下記の 4 つが重要なポイントとなる。

(a) 組織レベルでのアジャイル開発への理解と支援
(b) アジャイルな「思想」の正しい理解
(c) 各ステークホルダ間の円滑なコミュニケーションの促進
(d) 「単一工」型から「多能工」型人材への育成

(3) アジャイル開発が適さない領域

アジャイル開発手法が適さない領域もある。

Etsy は e コマースサイトの開発ではアジャイル開発、DevOps を導入し、成功したが、クレジットカード決済システムは PCI DSS 準拠のため、運用者の了承と実装が必須という制約があり、導入を断念した。

また、Huddle は小規模かつ徐々に機能変更を行う必要のあるシステム開発にのみアジャイル開発を採用し、大規模なシステムについては設計時点で仕様および要求事項を明確に規定することが重要であるとして、アジャイル開発を採用しなかった。

＜アジャイル開発や DevOps が適する領域＞

(a) 市場やニーズ等の不確実性の高いビジネス環境におけるシステムまたはサービス

＜アジャイル開発や DevOps が適さない領域＞

(a) 設計時点で仕様が明確なシステムやサービス
(b) リリース後の機能更新が少ないシステムやサービス
(c) PCI DSS への準拠等、開発者と運用者が明確に分かれる必要があるシステムやサービス

(4) マネジメントにおける覚悟

これまでの社内ルールや社内風土、慣習が阻害要因になるアジャイル開発は組織レベルでの理解と支援が必要であり、IT ベンダの経営幹部はマネジメントにおいて幾つかの覚悟が必要となる。

＜マネジメントに求められる３つの覚悟＞

(a) 失敗を成功の学びとする覚悟
(b) 完全性へのこだわりを捨てる覚悟
(c) 現場に自律と意思決定を委ねる覚悟

5.2 日本のシステム開発で求められる転換

(1) 高まるスピードへの要求

スタートアップやフロント系システムだけでなく、基幹系システムもスピードが重要になってくる。

たとえば、携帯キャリアが他社の新料金プランに追随したり、損保会社が他社の新型特約に追随するには、フロント系だけでなく基幹系や基幹に近いシステム開発のスピードアップも必要となる。

さらに、スピードアップを求められる上に、コスト増について顧客の理解を得ることは極めて難しい。

「スピード＝価値」を顧客に説明して必要な費用を確保するのは IT ベンダの課題だが、とにかくスピードアップを実現できなければ顧客との交渉の土俵にも立てない状況が確実に近付いている。

スピードを上げるためにはアジャイル手法や DevOps が解決策そのものと言える。スピードを上げることで「システム品質」が低下することは避けなくてはならない。アジャイル手法や DevOps の導入がシステム品質を下げることにはならないが、IT ベンダの品質保証部門は従来の品質保証の枠組みを変える必要があるかもしれない。今後の課題の 1 つである。

(2) アジャイル開発が日本で普及しない要因

5.1 は日本の IT ベンダにも役立つものであるが、アジャイル開発がかなり以前から日本に入ってきているにも関わらず、他国と比べて極端に導入が進んでいないことはスクラム認定者数からみて明らかである(図 2-5 参照)。

日本でアジャイル開発の普及が阻害されている要因は、(独)情報処理推進機構から報告書が発行されているので、参考にしていただきたい。

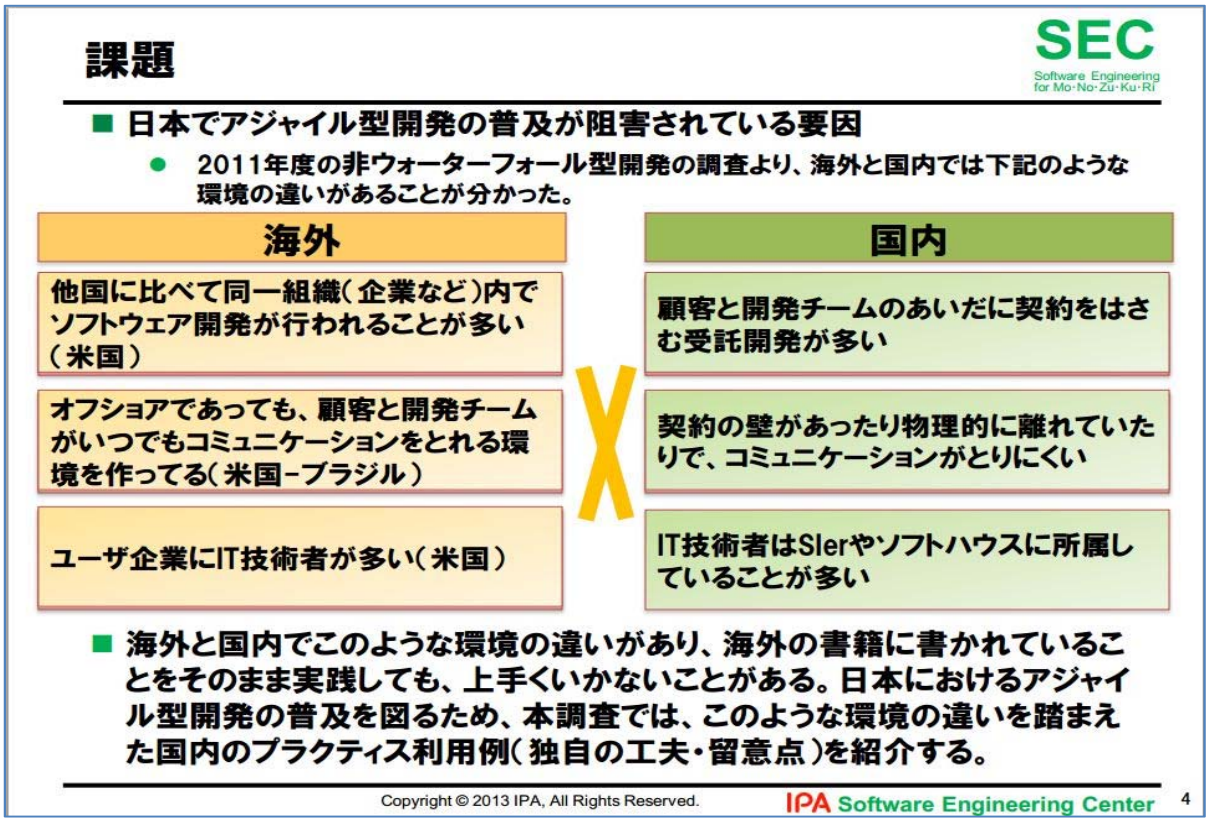


図 5-1　日本でアジャイル型開発の普及が阻害されている要因[1]

(3) クラウドサービスの活用方法

システム開発のスピードを実現するため、アジャイル手法やDevOpsの活用が有効であることを述べてきたが、それらは仮想化技術やクラウド技術、クラウドサービスをうまく活用することが必要である。

現在、Amazon Web Services(AWS)等、パブリッククラウドを利用することができるが、サービスの選定においては、「可用性」「性能」「拡張性」「運用・保守性」「セキュリティ」「機能性」「コスト」の観点を考慮し、プロジェクトの必要十分条件を設定した上で選定することが重要である。検討にあたっては、当WGが今夏発行する「クラウドインテグレーションにおけるSLA検討レポート(仮)」や経済産業省が公表している「クラウドサービス利用のための情報セキュリティマネジメントガイドライン」の活用も有効である。

[1] 出典 IPA, http://www.ipa.go.jp/sec/softwareengineering/reports/20130319.html

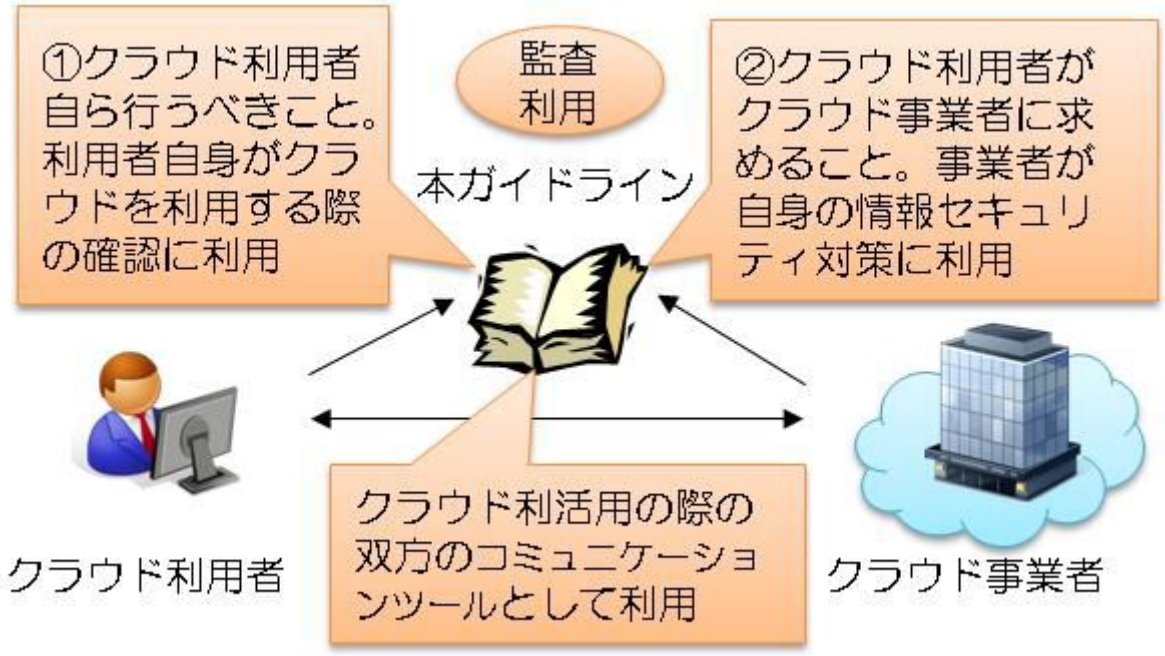


図 5-2　クラウドサービス利用のための情報セキュリティマネジメントガイドラインの活用イメージ[2]

(4) 最後に

最後に、本報告書の主張ポイントを下記にまとめる。

- クラウドサービスの活用促進により、SI 市場は縮小傾向でかつユーザ企業から厳しい要求を受けている
- IT ベンダは、開発コスト削減、短納期化、仕様変更に対する柔軟性、品質確保を実現しないと生き残れない
- 米国を中心にアジャイル開発、DevOps などリーンスタートアップでの成功事例がある
- 顧客からのフィードバックプロセスを組み込む、クラウドサービスを最大限に活用する、開発者と運用者が連携(DevOps)する、それが成功の鍵である
- 手法や技術の転換だけでなく「組織・文化の転換」と「経営者の覚悟」が重要である
- 日本の IT ベンダでもスピードが重要。アジャイル開発や DevOps の活用が解決の近道。日本ならではの課題はユーザ企業と連携して解決したり、セキュリティや契約等のガイドラインを積極的に活用しよう

[2] 出典 経済産業省,
http://www.meti.go.jp/press/2011/04/20110401001/20110401001.html

J25-9
クラウド時代の新しいソフトウェア開発の潮流
～米国の Agile/DevOps 適用事例からみる日本の IT 業界が求められる転換～

平成 26 年 4 月発行
発行：一般社団法人情報サービス産業協会
〒104-0028　東京都中央区八重洲 2-8-1　日東紡ビル 9 階
TEL　03-6214-1121
URL　http://www.jisa.or.jp/