SRA GROUP
GSLetterNeo Vol.118
2018年5月

アジャイルとは何か？

土屋 正人

◆はじめに

「アジャイル」という言葉がソフトウエア開発の世界で使われるようになって 20 年近く、日本で広まり始めて 10 年以上が経とうとしています。しかしながら、この言葉は未だに正しく理解されずに使われていることがあるように感じます。

「アジャイルって何？」という問いに

- ドキュメントを作らない
- 設計しない
- いつでも仕様変更に対応する
- スーパープログラマで作るチームだけが可能

という声を耳にすることがあります。これらは本当でしょうか？

◆誤解？事実？

- ドキュメントを作らない？

これは「ソースコードがドキュメントである」という視座、視点であり、その通りだと思いますが、「ソースコードをリファクタリングしていくことが前提」と考える必要があります。リファクタリングは、場当たり的なものではなく品質を改善するためのもので、そのためには設計原則の理解と実践が不可欠です。

例えば、代表的な設計原則のひとつに「単一責任原則（SRP: Single Responsibility Principle）」があります。これは、

- 変更する理由が同じものは集める
- 変更する理由が違うものは分ける

というものです。ソースコードは、リファクタリングによってこれらの設計原則に合致するよう改善されていくことで、ドキュメントとして機能するようになります。それによってクラスやメソッドの意図が明確になるかもしれませんが、

- 要求の理由
- システム要素間の関係性
- システムの全体像
- 業務とシステムの対応
- 業務のルールとその理由
- ビジネスと業務の関係

といったことは読み取れないかもしれません。これらが必要なものであるなら、形はどうあれ作る価値があります。

「いま必要なものを作る」のがアジャイルの視座だと思います。

- 設計しない？

「設計工程」を意識する必要はないかもしれませんが、先に記した設計原則を意識する時点で、すでに設計を行っているといえます。

- いつでも仕様変更に対応する？

開発量・作業量を無尽蔵に増やすことはできません。優先順位の高い要求が割り込む場合、優先順位

の低い要求とのトレードオフを行う必要があります。トレードオフは、広さと深さの双方から考えることができます。

- **スーパープログラマで作るチームだけが可能？**

改善を続け、成長していく。これができるチームだけが、アジャイルな開発が可能だと思います。

◆アジャイルは開発プロセスか？

アジャイルはウォーターフォールと比べられることが多いと思いますが、アジャイルは開発プロセスでしょうか？

アジャイルとは「アジャイルソフトウエア開発宣言 (http://agilemanifesto.org/iso/ja/manifesto.html)」にあるように、思想あるいは価値観というべきものだと思います。

ここで記されている 4 つの価値を最大限にすることが「アジャイルソフトウエア開発」であり、価値を最大にするためには、常に改善を意識する必要があります。

改善のためにはフィードバックする機会が必要で、フィードバックの間隔が短いほど改善機会は増えます。

ここから「短い期間で開発して素早くフィードバックを得る」という反復開発が芽生えます。

スクラムなどの開発フレームワークで定義されているスプリント（イテレーション）は、フィードバックを早く得るための施策のひとつです。気付きを早く得るという意味では、ウォーターフォールでもカンバンなどの施策を導入することで同様の効果が期待できます。

アジャイルとウォーターフォールはレイヤが異なります。比較するものではなく、融合するものでしょう。ウォーターフォールでも、例えばカンバンなどを使ってアジャイルな開発は可能だと思います。

◆不確実性

予測精度が低いものは不確実性が大きくなります。予測精度が高いものはすぐに着手できますし、予測との誤差も小さくなることが期待できます。天気の長期予報と週間予報の違いのようなものでしょう。

図 1 は不確実性コーンと呼ばれるもので、プロジェクトの初期段階では見積り誤差が大きく、プロジェクトが進行する――すなわち確定したものが増えるに連れて不確実性が減ることを示しています。

自明なことかも知れませんが、初期段階での見積りがそのまま使われて不確実性が減少した時点でも再見積りを行わない、というプロジェクトが少なからずあるのではないでしょうか。

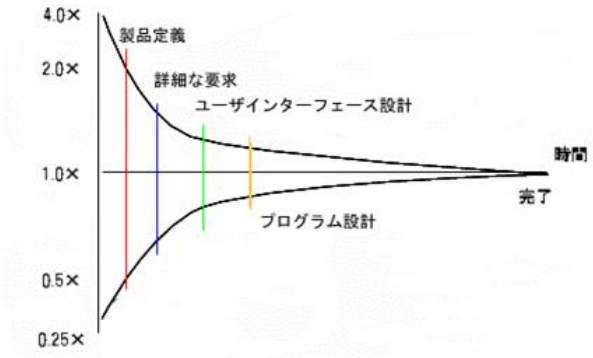

図 1　不確実性コーン

◆どうすれば予測精度を高めることができるか？

まず、顧客がすぐに欲しいと思っているものから着手することが挙げられます。すぐに欲しいということは、具体的な利用シーンと理由を持っているはずです。これをストーリーと呼び、カードに書き出します。

次のようなテンプレートが知られています。

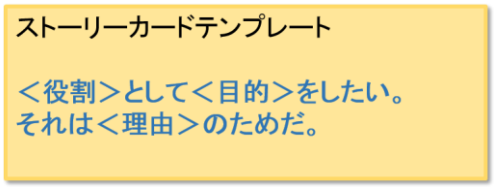

図 2　ストーリーカードテンプレート

例えば、次のようなものになります。

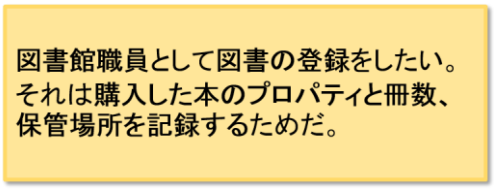

図 3　ストーリーカードの例

ストーリーを集めることができたら、プランニングポーカーを行うことが有効です。これはストーリーを相対的に見積るもので、見積りと同時に業務の理解、要件の理解、設計要素の識別などを共有することができます。やり方については Vol.3 を参照してください。

プランニングポーカーは時間がかかりすぎる、という批判がありますが、単なる見積り作業と捉えるからそう思えてしまうのであって、業務理解、要件理解、設計要素の識別を同時にやっていると考えれば、そのような批判は当たらないと思います。

開発が始まったら、チームの開発速度（ベロシティ）を知ることが有効です。チームのベロシティを計測することで、残りの開発に必要な時間を予測できます。ただし、そのベロシティはチーム固有のものです。他のチームに適用したり、一般化したりすることはできません。

スプリントにしても、プロトタイピングにしても、早めに失敗して、小さく成功します。早めの失敗から得られる改善策と小さな成功から得られる達成感が、不確実性を減らして予測精度を高めます。

一般化して言えるのは「コントロールできるものとできないものを知ることが大切」ということです。「コントロールできないもの」が不確実性を高め、予測精度を下げます。「コントロールできないもの」を何とかしようとしても、不満や不安が高まるだけです。

他人の気持ちをコントロールすることはできませんが、自分の振る舞いをコントロールすることはできます。

「コントロールできるもの」を工夫することで、「コントロールできないもの」に対応することが、できるかもしれません。

GSLetterNeo Vol.118
2018 年 5 月 20 日発行
発行者●株式会社 SRA 先端技術研究所
編集者●土屋正人

バックナンバを公開しています●http://www.sra.co.jp/gsletter
ご感想・お問い合わせはこちらへお願いします●gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋２－３２－８

夢を。Yawaraka Innovation
やわらかいのべーしょん