ソフトウェア開発の「見える化」
～リーンソフトウェア開発～

株式会社永和システムマネジメント
平鍋　健児

ソフトウェア開発現場で、よくこんな声を聞きます。

- 現在全体のどれくらいが完成しているのか、よく分からない。（マネジャ）
- 設計が90%完了のまま、２週間同じ報告が続いている。（マネジャ）
- プロジェクトは遅れているということだが、自分は今日何をすればよいのか分からない。（開発者）
- プログラミングが大好きでこの業界に入ったが、毎日、仕様書づくりと指示どおりの実装や保守作業に追われて、仕事が楽しくない。（開発者）

このような状態は、この業界ではかなり典型的な症状でしょう。管理と開発の分離が進みすぎた場合や、管理と開発の報告インターフェイスが良くない場合、チームのコミュニケーションが良くない場合、過度な役割分担による作業の分業が進みすぎた場合にこのような状況が起こります。ここでは、この症状の打開策として、「見える化」（目で見る管理）の重要性と、実践方法を考えてみたいと思います。

■まず、目に見えるようにせよ　～見える化

最初に大切なことは、プロジェクトの状態をとにかく見えるようにすることです。マネジャによっては、「進捗はエクセル（もしくは○○Projectツール）で管理しており、それを毎週メールで配布して、メンバーや上位のマネジメントと共有している（あるいはWebで見えるようにしている）」と言う方もいるでしょう。私は、この方法はある程度効果があるものの、むしろ問題が多い、と考えています。

図１　壁一面の「貼り物」

メールで送られている情報は本当に見られているのでしょうか？添付ファイルで送られた進捗状況を開いて、メンバーは確認するでしょうか？自分の抱えている問題との直接の関係を見つけにくく、見ても有用な行動を起こせない、ということはないでしょうか。上位マネジメントから、「忙しいので細かい資料を見ろといわれても困る、結局結論としてどうなのか説明して欲しい」、などといわれたことはありませんか？

私は、プロジェクトリーダーに「何でもよいので進捗を示していて、かつ、毎日変化するものを壁に貼ってください」という指示をするようにしています。複数のプロジェクトがあると、この「貼り物」が会社のフロアー一面に貼り出されます（図１）。毎朝、出社すると各プロジェクトの貼り物が一斉に目にはいります。マネジャはそこでライブな情報を歩きながら得ることができます。

貼り物の良いところは、それが自分で情報を発信していることです。そこを通る人が、それを目にするだけで情報を得ることができます。忙しいマネジャは、パソコンを立ち上げてメールを見、添付ファイルを確認する、という面倒な情報の取り出しには反応しません。情報を壁に張っておけば自然とそれは目に入ります。さらに、その情報が開発の現場（現地）にあることが重要です。分からないことはその場で質問することもできるのです。

また、開発者も、自分たちの状況がそのまま壁に貼りだされるため、毎日同じ情報を全員で共有することができます。「あの人は完了だと言っているが、この人はまだ50%くらいだと言っている」というような曖昧性はありません。その情報が「正」として、そこに張り出されています。毎朝、開発チームはこの貼り物の前で、短い「朝会」をスタンドアップミーティング[1]の形式で行ないます。現状を確認し、今日の作業を確認することで、自分で行動を起こすことができるようになります。メールによる伝達では、問題点についての会話を誘発することができません。貼り物を前にしたミーティングでは、今、見えている状況を見ながら、問題に対しての議論が自然と起こってきます。このフィードバックが、見える化の大きな力となって、開発者を自発的にドライブするのです。

■全体進捗の見える化　～「バーンダウンチャート」

では、何を情報として張り出すのでしょう。まずは、「進捗」という観点から考えます。進捗管理の基本は、「中間生産物で計測しない」ということです。「設計が90%完了のまま、２週間同じ報告が続いている。」というような問題は、ゴールが不明確であることから起こります。「明確に０か１かで判断できる計測単位であること」、さらに、その単位は、「顧客価値と直結していること」が重要です。「設計書のページ数」という単位は、２つの条件を両方とも満たしていません。ページ数の予定と実測は、設計書の完成というゴールを明確にしていませんし、さらに、おそらく顧客は設計書の完成に価値を感じていないでしょう。設計書は「中間生産物」の代表例です。

筆者が提案する最もよい進捗管理の単位は、「受け入れテストを通った顧客要求の数」です。テストは、０か１かで成否判定ができます。また、顧客は動くソフトウェアを求めており、受け入れテストをパスした、ということは顧客価値に直結しています。顧客が求める仕様に対して受け入れテストを定義し、これを通過した数で進捗を計ります。読まれない仕様書、テストされていないソースコードなどの中間生産物は製造業における「在庫のムダ」に相当します。このような「中間状態にあって価値に直結しないもの」で計測してはいけません。

テストを通った顧客要求の数を進捗の単位にするには、ウォーターフォール型のプロセスでは難しいかもしれません。全体を分析、設計、実装、テスト、という工程で流していく手法では、テストをパスする数は工程の最後になってしまいます。繰り返し型のプロセスを導入する、もしくは、要求全体を分割して、少しでもいいから優先度の高い要求を最終工程にまで早く流してしまうことを考えましょう。そうして、最終工程である受け入れテストを通過した要求の数で、進捗を測るのです。

---

[1] 朝、全員参加で立ったまま行なう15分程度の会議。一人ひとりが、昨日までの状況、問題点を簡潔に話し、今日の作業予定を明確にする。

アジャイル開発では、要求をストーリーという顧客価値の単位に分割し、それを「一個流し」します。全体を見据えた大きな分析・設計に時間をかけずに、小さな要求単位をテストにまですばやく流す、ということを繰り返します。この手法は、優先度の高い要求から顧客に早いうちから供給することができるため、最初に全要求を固定する必要がなく、都度の要求変化に強いプロセスでもあります。

図２に、「受け入れテストを通った顧客要求の数」を指標にした、「バーンダウンチャート(burn-down chart)」の例を示します。

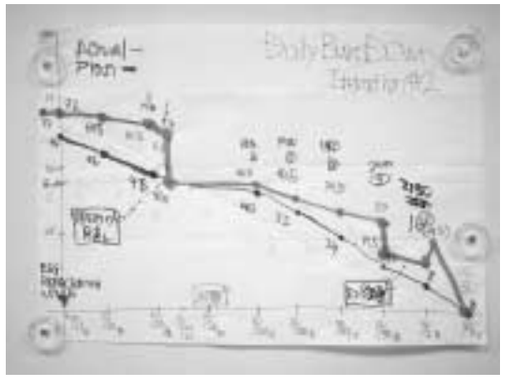

図２　「バーンダウンチャート」の例

このチャートは、１つの開発期間内に予定している「顧客要求の数」を「受け入れてテストを通った数」だけ引き算していき、「今残っている顧客要求の数」を日ごとにプロットしたものとなっています。つまり縦軸は残作業量です。まず予定線を引き、それに対して日々の実績線を引いていきます。開発期間の終了に向けて、開発の収束度合いが一目で分かるという特徴があり、このカーブの傾きから開発の終了予想ができます。このカーブを日々見ていくことで、問題の早期発見と対策に向けたアクションを取っていくことができます。

■日々の作業の見える化 ～「ソフトウェアかんばん」

バーンダウンチャートは、全体進捗を可視化しますが、これだけでは日々の個人の作業が明確になりません。顧客要求を開発作業にまでブレークダウンし、それをチーム内の個人の作業として明確にする必要があります。この基本は、要求を作業項目に落とし「担当者」と「期日」を明確にすることです。アジャイル型開発では、この作業単位を「タスク」と呼びます。顧客要求を元にブレークダウンされたタスクは、カードに書いて担当と期日を明確にして、貼り出します。これが「ソフトウェアかんばん」です(図３)。

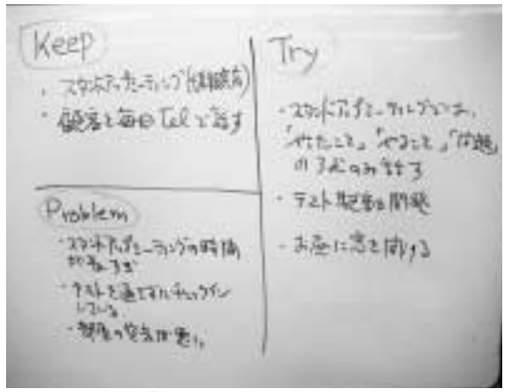

図３　「ソフトウェアかんばん」の例

この貼り物は、「ToDo」(未実施)、「Doing」(実施中)、「Done」(完了)の３つの欄に分かれており、最初、すべてのカードはToDoの位置にあります。担当者が決まると、担当者はこのカードに自分の名前と終了予定期日を書き込み、Doingの位置に動かし、作業を開始します。そして、終了するとカードに終了日を書き込み、Doneの位置に動かします。１つの開発期間に渡って、全カードがToDoからDoneに向けて移動して行きます。この「ソフトウェアかんばん」を使って、チーム内部の日々の進捗を管理するとともに、各人の作業を明確にすることができます。

タスクの担当決めで重要なことは、「自発的サインアップ」です。日々の朝会の時に、「私がこのタスクをやります」と宣言してカードに名前と終了予定を書き込むのがよいでしょう(図４)。

マネジャもしくはチームリーダーは、できるだけ細かい指示・指名をせず、自発的なやります宣言をするムードを醸成します。これは、チームづくりにおいて最も大切にしたいことです。細かい作業を割り当てるスタイルの管理を続けていると、朝、開発現場に出社して、「さて、私は何をやったらよいのだろうか。指示を待っていよう」というチーム文化を作ってしまいます。「ソフトウェアかんばん」と「自発的サインアップ」、「朝会」をうまく使って、毎朝出社したら、現在チームがどういう状態であり、自分はどういうタスクを受け持っており、何をすべきなのか、という自己判断ができるようにしましょう。シンプルな見える化と、自発的なムードによって、マネジャやリーダーの介在を減らしつつ、作業の流れがスムーズになることを実感できるはずです。

図４　朝会（スタンドアップミーティング）

■アナログとデジタルツールの使いわけ ～ツールが得意なこと

これまで、「見える化」をキーワードにして、２つのアナログツールを使った可視化手法を見てきました。ソフトウェア開発をしていると、ついついデジタルツールに頼ってしまいがちです。しかし、創造的な開発現場では、アナログツールの視認性とスピードが勝ることが多いようです。

逆に、デジタルなツールが有効な場面もたくさんあります。例えば、繰り返し行なう作業はすべて自動化することを考えましょう。例えば、ソフトウェアのビルド、テスト、構成管理、状態遷移表からのコード自動生成、など、決まった作業、膨大な作業、繰り返しおこなう作業はすべてツールを使って自動化してしまいましょう。特に、日々のビルドと回帰テストを自動化することは重要です。筆者が以前たずさわったプロジェクトでは、最新版の全ソースコードとテストコードをリポジトリからチェックアウトし、ビルドし、回帰テストを走らせ、その結果をプロジェクトのメンバー全員にメールで送信する、という一連のバッチプログラムが、一日に２回、自動で動くようになっていました。

また、UMLを使って設計を行なう場合、最初はホワイトボードでかまいません。段々量が多くなったり、整理したりする必要がある場合には、やはりツールを使うのがよいでしょう。ここでは、Jude（ジュード）という私も開発に関わっているUMLツールを紹介しておきます[2]。

特に、UMLはクラス図の静的な情報とシーケンス図や状態図などの動的な情報が連動すると、

2　Judeはフリーで配布されており、年間20万ダウンロードのアクセスがあります。
http://ObjectClub.esm.co.jp/Jude/
ちなみに、この記事で紹介した「ソフトウェアかんばん」や「朝会」の写真はJudeの開発チームのものです。

非常に便利です。PowerPointやVisioで書くよりもすばやく一貫性を持って書けますし、修正がとても楽になります。筆者は、基礎となるモデルが固まったら、A2くらいの大きな紙にUMLのクラス図を描いて壁に張り出すのが好きです。最近は、カラープリンターが普及してきましたから、特に色をつけてクラスの役割を分類すると、分かりやすく印象に残りやすい図が描けます。

基礎となるモデルのUMLクラス図を壁に貼っておくと、設計に関するメンバー同士の意識あわせや問題解決が、自然とその図の前で始まります。

■アジャイル開発と見える化 ～「リーンソフトウェア開発」

さて、ここまでソフトウェア開発の「見える化」に焦点をあてて、その有効性と具体的な実践方法を書いてきました。読んでいただいて分かるとおり、これには多くの「チームビルディングの視点」が含まれています。そして、ムダの排除、改善、自発的な組織づくり（人間性の尊重）、目で見る管理、ジャスト・イン・タイム、と言ったトヨタ生産方式をはじめとする「生産革新」の現場で行なわれてきた実践項目と符合しています。アジャイル開発を、この視点から捉えなおした新刊書籍が、Mary Poppendiek / Tom Poppendiek著の『リーンソフトウェア開発ーアジャイル開発を実践する22の方法』(拙訳)です。

アジャイル開発は2000年に「XP（Extreme Programming)」というプログラミング技法、テスティング技法、チームづくりの究極的な体系化としてセンセーショナルに登場しました。この時点では、プログラマーの復権宣言のようにも解釈されることがあり、物議をかもしましたが、2002年、他にも同様の価値観をもつ方法論の総体を特徴づける「アジャイル宣言」[3]が発表され、Scrum/FDD/DSDM/Crystal/ASDなどのアジャイル開発方法論が一つのムーブメントとして注目されるにいたっています。Jim Highsmithは、「複雑系」の理論をもちいてこれらの手法が現代の変化に富んだビジネス環境に適応するための手法であることを明らかにしましたが、「本当にうまくいく」ということを実証する最後の課題は残されたままでした。今回の

[3] http://www.agilemanifesto.org/

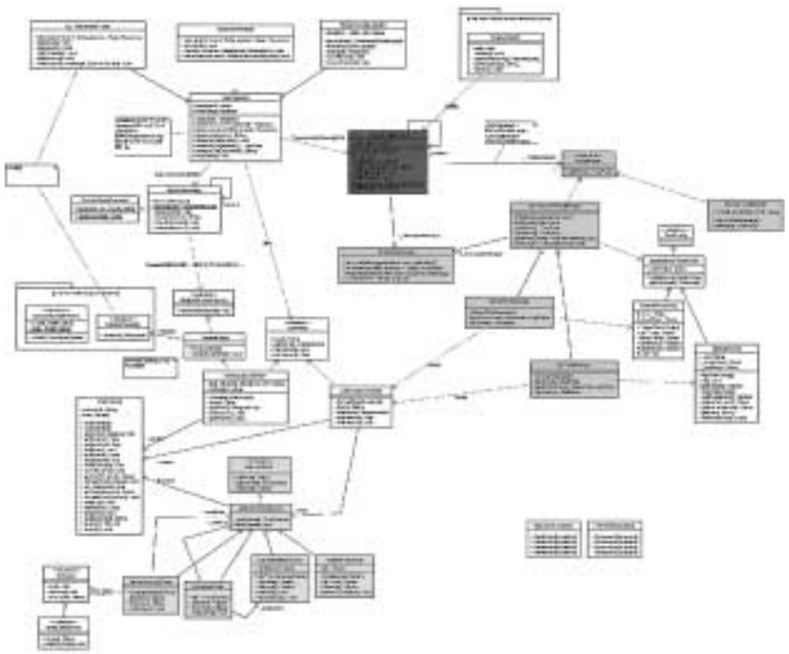

図５　UMLクラス図の例（Judeによる）ーこれも壁に貼る

『リーンソフトウェア開発』は、過去20年間に渡って製造業の現場で形作られ、実証されてきたリーン・プロダクション・システム（トヨタ生産方式を含む）の原則を用いて、アジャイル開発の有効性を解き明かしたものです。

図６　『リーンソフトウェア開発』日経BP社

この本では、７つのリーン原則から、22の思考ツール（考える道しるべとなるキーワード）を導きだして解説しています。本書の主張は、「プラクティス（実践項目）は他の場所でうまくいったものや、本に書いてあることをそのまま行なってはうまくいかない。その背後にある原則を理解し、自分の環境に合わせて、プラクティスとして編み出さなくてはならない」というものです。そのプラクティスを編み出す過程として、７つの原則が有用なのです。以下に、７つの原則を簡単に紹介しましょう。

原則１　ムダを排除する

ムダ、とは顧客にとっての価値を付加しないもの、すべてである。ソフトウェア開発における７つのムダ（未完成作業のムダ、余分なプロセスのムダ、余分な機能のムダ、タスク切り替えのムダ、待ちのムダ、移動のムダ、欠陥のムダ）を発見し、ムダを排除しよう。

原則２　学習効果を高める

ソフトウェア開発プロセスは、繰り返し可能な「生産」ではなく、常に「発見」を繰り返す「学習活動」である。この学習プロセスを機能させるために、活動を見える化し、フィードバックを得ながら自己を改善していく仕組みを作ろう。

原則３　決定をできるだけ遅らせる

不確定要素が多い場合、確実な情報を元に決定を下せるように、「オプション」を維持したままで前進することを許容しよう。このためには、システムに変更可能性を組み込んでおくことが戦略的に重要である。

原則４　できるだけ早く提供する

「完璧主義」に陥らず、とにかく早く提供する。顧客からフィードバックを得ることで、発見と学習のサイクルが生まれる。このためにも、顧客からのプル型で、開発を進めよう。

原則５　チームに権限を与える

現場の開発者が、100%の力を出せるようにする。中央集権で管理しようとしてはいけない。自発的な決定ができるようにチームをモチベートする。見える化の手法をうまく使って、チームが自分の意思で状態を確認しながら前進できるようにしよう。

原則６　統一性を作りこむ

統一性が感じられるシステムには、一貫したビジョンと思想がある。これはプロセスや手順で作ることができない。リーダーシップとコミュニケーションが、統一性の泉源となる。

原則７　全体を見る

部分最適に陥ってはならない。個人や一組織のパフォーマンスのみで評価すると、部分最適が起こってしまう。一つ上のレベルで評価するようにし、個人や組織の協調が生み出されるようにする。

■まとめ

ソフトウェア開発における、「見える化」をテーマにして、その意味を明確にし、方針と、具体的な方法を紹介しました。そして、自発的なチームのマネジメント方法についても少し触れ、同じテーマの新刊書籍『リーンソフトウェア開発』を概説しました。

みなさんも、早速自分の開発現場にかえって、まずは見える化から着手してみませんか？きっと新しい発見があるはずです。

以上