現場のジレンマと解決 ～ 見えないものが見えてきた ～

松下電器産業(株) AVC 社 ADD 商品技術部　根岸 政人

■ 非論理の世界から

昔あるところに、上司の指示をよく守る部下がいました。

「仕様書作成に時間かかりすぎじゃないのか?」

「日程を見直すべきです。仕様を全部書くのにもっと時間が要ります」

「詳しい部分はソフトハウスの仕事だろ。こんな機能がほしいと書いて、あとは任せて早く設計にかかれ」

若かった部下は、仕様書は設計じゃないのかと疑問を持ちながらも、上司に弱いサラリーマンのサガに従いました。結果は、バグが多発して修正に修正を重ね、検査も行き届かず、バグのモグラたたき状態になって、てんてこ舞いしました。品質管理部門に行くと「また君の季節になったか」と妙に歓迎され、マイコンソフトの切替日程調整で工場に頭を下げ続けました。

「どうして仕様書に書いてないんだ」

「指示どおり、詳細はソフトハウスさんに任せました」

「こんなことは仕様書に書いておくことだろ。それに、何をレビューして何をテストしたんだ」

「仕様書をレビューして、仕様書に基づいてテストしました。でも日程が押していましたから、仕様書には機能説明しかありません」

「日程より品質優先が当たり前だ。日程どおりできないなら報告するのが部下の役割だろ」

「あのとき言いました！」

上司の言うことを聞かず我が道をゆく部下が、こうして生まれました。

部下の反抗の日々は続きます。

「センサ 2 個では 4 種類の状態しか判別できません。それ以上の状態があるんですから、時間軸上のタイミングを利用するか、センサを増やすべきです」

「寸法ばらつきがあるからタイミングは保証できんし、コストアップは許されん。何とかならんか」

「何ともなりません」

「まだ動かしてもないのにやな、やってみなわからへんやろ。頑張りが足らんのやないか」

「やるまえにわかります。頑張ってできるんもんじゃありません」
「君は仕事になると堅いな。堅いこと言わずに、どうにか、な」
「どうにかとおっしゃられても、論理的に明らかなんです」
「しゃあないな、センサを増やすから、そのコストは電気回路で削減してくれや」
「え、それ、どうして電気のコストに振るんですか！」
組み込みマイコンのターゲットは、非論理の世界からやってきます。

新しいターゲットを動かすのは技術的にはとても面白いのですが、「やってみて決める」の間は、多人数でのシステマチックな開発になりません。

■ 見えない

● 何を作るかが見えない

「この機能は仕様書に解説しているように動作します。あとは、使い易くて不具合のないようにお願いします」機能概要だけを書いた仕様書を渡し、それを受けていきなりソースを打ち込む、以前はこれが可能でした。

バグを振り返ってみますと、バグの多くは、想定していなかった条件で発生しています。組み込みシステムにおいては、ターゲットをよく知っているのは仕様作成者です。ソフト設計者にすれば、ターゲットとなるハードの動作がよくわからないままに仕事を任されてしまいますので、「こんな状態があったのか！」と驚いたりすることになります。その結果、「仕様書の抜けだ」「いや、ソフト設計の抜けだ」と責任のなすり合いになります。

新しい製品をつくるときは、いつでもチャレンジしていますから、やってみなければわからない部分もあります。ひととおり設計完了して実行したら、とんでもないバグが発見され、作り直そうとしても担当者以外のメンバーは何もできず、「いつできる?」と、リーダーは担当者にプレッシャーをかけるだけになったりします。

ここでの課題は、次の 3 点です。
(1) 仕様書にある「抜け」を発見できない。
(2) 機能を実現するために何を作ったらいいのかイメージできない。
(3) 作るソフトと仕様書の関係が担当者以外には理解できない。

仕様書に抜けがあるのは、仕様作成者が表現方法を知らないことと、組み込みマイコンの規模が大きくなるにつれて、全体像も詳細部分もと

らえにくくなるからです。

例えば文章だと、わかりやすく書いたつもりでも相手の受け取り方が違う場合があります。文学が面白いのは含みがあるからで、論理的な表現には向きません。私も、他人に伝えることの難しさに、自分の能力を嘆いたときがありました。できるかぎり詳しく書こうと取り組んでも、時間がかかるばかりで頭が飽和して、かけた時間だけの成果が出てきません。

そんな仕様書を元にレビューするわけですから、大勢の参加者を募って経験者の気づきを文章で追加しても、根本的には変わりません。バグの反省として「もっとレビューを」と結論づけることがありますが、そうは言っても、どんな資料を作ったら他人に理解されるのかわからないのです。レビューに時間をかけたところで何も解決しません。レビューの資料や手法に問題があるからです。

仕様書は、仕様作成者の頭にあるイメージをソフト設計者に伝える手段です。ターゲットの動きがイメージでき、ソフト上の実現方法がイメージでき、作るソフトと1対1に対応していて他人にも理解できるのが、理想の仕様書でしょう。

● 工数が見えない

組み込みマイコンの多くは、完成日が先に決められます。クリスマスセールや入進学に合わせた商品の発売タイミングがあるからです。

設計に着手したとき、漠然と「やることがたくさんある」と感じられます。機能仕様書とターゲットを前にして、やるべき作業の量が見えてきません。工数を見積もるために、とにかく必要な関数をあげようと取り組んでも、全ての条件を見切れているかどうかの不安がずっとつきまといます。工数の見積もりが立たないので、とにかく早くコーディングにかかれと指令が出ます。システム分析が未完のままコーディングすると、テスト工程で抜けが発見され、開発の後戻りになります。

さあ、仕様変更が発生しました。どれだけの期間で対応できるのでしょう。1週間で作って1週間でテストします、と大雑把に答えが返ってきます。個人の能力に頼っていたら、その期間を決める根拠は過去の経験だけです。変更すると、その変更が及ぼす影響の範囲が見えない、だから変更に要する工数も見えません。

メカニズムや半導体を取り付け、「明日、とりあえず動くようにして見せ

てくれ」と言うボスもいて、呆れることもありました。「とりあえず」のために工数が割かれ、後の仕事を圧迫しますので、日程調整が必要になります。

ビジネスライクに進めるためには、品質とコストを踏まえた工数見積もりの根拠が必要です。

● 成果も見えない

最初は仕様書がありますが、途中で仕様変更が入るのが常です。設計を進めていくと、仕様変更を仕様書に反映させる時間がとれなくなって、最後にはソースファイルだけが正しいことになります。商品展開や次期マイナーチェンジのためにドキュメントを残そうと思っていても、開発完了したらすぐに次の仕事がありますから、書く時間はありません。成果物としてのドキュメントを要求されながら、作成する時間を確保できないのです。そうすると、努力しても成果として評価されるのはバグの件数だけ、という減点主義になってしまいます。

新製品に多くのチャレンジをするとその分バグも多くなり、工場や修理部門に何度も頭を下げにまわる羽目になります。そしてエンジニアはだんだんチャレンジしなくなり、製品は弱くなっていき、売れないから給料が減っていきます。

組み込みマイコンのソフト設計者の成果がなかなか認められないのは、見える形で残しにくく、残しても再利用しにくい場合が多いからです。自分の成果は、と考えても見せられるほどのものがありません。成果は頭の中、というのでは個人の能力に頼っている状況から抜けられません。

■ そして状態遷移表

● 仕様作成者とソフト設計者をつなぐ -適用事例-

ビデオ CD チェンジャ SL-VM535 の開発を機会に、ZIPC を購入しました。弊社製品は以前から 78K/0 シリーズのマイコンを搭載しており、idea-L を使用して開発を行っていました。新製品に新メカニズムを採用し、さらなる機能アップを高品質で実現するため、idea-L と ZIPC の両方を使って、今回は特に NEC マイコン事業部殿および関連会社殿にご協力いただき、開発にチャレンジしました。

この商品では、ひとつの開発の中で成功と失敗の両方を経験しました。

ZIPC で状態遷移表化したメカニズム制御部は、仕様変更に即時対応

しました。変更の影響範囲が明らかなので、迷いがありませんでした。さらに、状態遷移表をテスト仕様として用いてデバッグした結果、その後のバグはゼロになり大成功でした。

これに対し、ユーザーオペレーションを文章で記述した部分は、200 件を越えるバグが発生しました。ひとつつぶせば副作用が起き、ちょっとした条件の違いでまたバグが出て、収拾のつかない状態になりました。それは個人の資質の問題ではなく、従来型の設計でカバーしきれない、規模の限界に突き当たったからです。

そこで、ユーザーオペレーション処理の中でいちばん重要な部分をZIPC で状態遷移表に作り替えました。ZIPC の状態遷移表をソフト設計者にメールし、レビューを行って完成させた結果、その部分だけはバグが皆無になりました。状態遷移表化に工数が増えましたが、もしそのまま続けていたら、後のデバッグには予定外の膨大な工数がかかったでしょう。最初から状態遷移表で、地道にやればよかったのです。

ZIPC の状態遷移表で、仕様作成者とソフト設計者の意志疎通ができました。文章中心の仕様書を提示して、行間を読むようなあやふやなことが全くありません。たとえば、仕様書として状態遷移表を作成し、各升目の中にソフトを組み入れてもらう、といったことが可能です。ただし、仕様作成者の頭の中がきちんと整理されている必要があります。口頭だけで伝えようとする半端な技術者は排除されていくでしょう。本当にいいことです。

状態遷移表のレビューは全員参加型です。今回の製品では実際に7名でレビューを行いました。ある機能を実現するために何をしたらいいのかを、全員が認識することができます。状態遷移表を通してターゲットの姿が見えるので、仕様の抜けを誰もが発見できます。全員がほぼ同じレベルで議論できますから、誰が決定権を持つのかを明確にしてレビューを実施すべきでしょう。このレビューに参加することによって、新米エンジニアも段取りを学び、設計が速くなります。ひとりひとりがエンジニアとして立っていくためのスキルを磨くことができます。

状態遷移表のテストは、升目をひとつひとつ通すことですから、作業内

容が非常に明確です。さらに、製品開発サイクルを短縮するためにテスト項目を絞る場合には、製品ユーザーの使用頻度が高い項目だけを抜き出すことも容易にできます。そして、升目の個数でテストの進捗管理をしてしまいます。シミュレーション段階では、ZIPC のカバレッヂ機能を使って管理することも可能です。

品質と日程管理を妨げる、設計効率を上げられない、保守管理の工数が大きいなどの最大の要因は、仕様書です。機能仕様書から設計内容を表現し、仕様作成者とレビューし、そのまま設計でき、そのままテストに使え、論理的に仕様変更がなされ、ビジネスとして管理ができる、そんな世界を状態遷移表とZIPC と、それを使うエンジニアが作ります。

● 目標が見える

私の趣味のひとつはマラソンです。一流のランナーはスピード競争に入っていますが、一般市民ランナーはまだまだ忍耐と根性の世界です。大会当日、最初は 5 キロごとのラップを目標に、軽快に走っています。30 キロを過ぎたあたりでグリコーゲンが枯渇してしまい体が動かなくなって、ゴールが限りなく遠く感じられてきます。目標が見えないのは、つらいことです。そんなとき私は、目の前に小さなゴールを決めます。「次の給水所まで行こう」、「あのカーブまで走ろう」と思うのです。その繰り返しで、最後には本当のゴールに行き着くことができます。小さな目標に分けてしまうことで、大きな目標が達成できます。

状態遷移表を作ると、作業はひとつひとつの升の中です。仕事は細分化され、具体的な作業内容が明らかになってきます。作業が具体的ですから、メンバーへの指示も具体的になります。各メンバーの能力に応じた配分が可能になりますから、チームの持てる力を有効に使うことができます。そうすると、時間の見通しが立ってきます。各メンバーからの報告も、升目の数でやってしまうとか、大胆に言えば升目の数で進捗管理もできてしまうのです。さらに、小さな目標が間近にありますので、「これをやったら仕事がひとつおわる」と、メンバーのやる気が出てきます。

組み込みマイコンの設計は、抽象的な製品イメージをいかに具体的な仕様書に表現し、いかに抽象化して

設計するかが、大切なポイントです。状態遷移表がそれを可能にします。階層化することによって、全体も部分もとらえられるようになります。状態遷移表は、仕様作成者とソフト設計者をつなぐ共通の言語です。ターゲットイメージが明確なので、両者が一堂に会してレビューが可能ですし、だから管理ができるようになります。両者の関係が活性化され、ますます協調性が高まり、加速度的に仕様書の完成度が高まっていきます。

楽しく仕事をしたい、それには、技術的にも管理面でも見通しがきくことが大事です。目に見えなかった品質と工数を見えるようにする、曖昧な不安を具体的な目標にしてしまう、これが ZIPC の魅力です。

● 状態遷移表は無敵か

以前、CD チェンジャのメカニズムを制御するソフトを設計したとき、作っていったら構造化設計になり、オブジェクト化して、オブジェクトの中は状態遷移表になっていました。そして私には、状態遷移表で設計する意識が生まれました。

状態遷移表は、教科書にあるような理論モデルではなく、もっと実践の修羅場に近い手法です。それを工夫して実用化した ZIPC は、開発の最前線で使える非常に優れたツールです。

じゃあ状態遷移表は万能で無敵でしょうか。それは違います。状態遷移手法に適さないターゲットもあるからです。複数で有限の状態と複数のイベントが存在して、遷移していく場合には最適です。順番に処理していくだけのものにはフローチャートやデータフロー図が適当でしょうし、手法を問わず速さを求めるアルゴリズム系は、ちょっとオタクな人にお任せするのがいいでしょう。ZIPC を定着させるには、どんな部分に状態遷移表を適用するか判断できる能力を育てることです。

「自分で使ってみろ」と私はよく言います。自分で体感しない口だけの評論家を、ビジネスパートナーには選びません。マニュアルを見ないと使えない、というのでは普及しません。イメージどおりに動いてほしいのです。ツールの使い方に気を取られないこと、ターゲットに集中できること、存在を感じさせないツールであることを、ZIPC をはじめとする全ての開発ツー

ルに望みます。

● 導入は難しい?

「ベルリンの壁」は、まず上司の意識です。組み込みマイコンの規模がどんどん大きくなってきて、管理者の人たちが開発していた頃の 10 倍以上になっています。「10 年前、私はこうして成功した」と言ってはばからない人がいますが、ソフトウェア開発の急激な需要拡大と開発方法の変化で、5 年前、10 年前の仕事のやり方は通用しなくなっています。あまりにも生産効率が低いからです。単に経験と慣れだけで仕事が速い上司は、すぐに抜かれます。生産性の競争です。生産性を上げられないところは淘汰されていくでしょう。

従来製品のマイコンソフトを、いかに早く完成させるかも課題です。ソフトハウスに、工数半減の取り組みをお願いしています。多くの目利きにレビューしてもらうとか、評価用ターゲットを増やすとか、そんな人海戦術で工数半減は不可能です。日の丸特攻隊が勝つ時代は、とうに終わっています。市場の要求は、急激に大きくなってきています。だからこそ、整った手法が必要なのです。

新しいツールの導入は、困難なことです。すぐには成果が出ませんし、かえって工数が増えるように見えます。また、状態とイベントを分析して区別する訓練が必要です。多くのソフト設計者は、ある程度のイメージ能力を持っていますから、少しの時間で訓練可能です。

訓練は、初心者をある程度の能力まで引き上げる効果があります。ソフトハウスから新人が来たりして、「うちは教育機関じゃない」と言いながらも育てるわけですが、いくらOJTとはいっても適切な手法がなければ育ちません。また、ソフト開発に 35 才定年説がいまだに言われるのは、頭の柔らかさにだけに頼っているからです。状態遷移表の手法は、そんな年齢の枠を取り払ってくれるでしょう。

ZIPC は、やり抜けば完璧を与えます。しかし、全体に一度に適用するには無理があります。そこで、ターゲットの最も重要な部分について、状態遷移手法を導入する手があります。少しずつ使ってみるのです。じわじわと成果が現れ、3 年後には組織的に使っていることでしょう。

導入するために最後に必要なのは、情熱を持ったリーダーの存在です。

周りの意識を変え、新たな開発手法を導入するには数年の時間がかかります。それだけに、柔軟な思考と強い意志を備えたリーダーのもと、駅伝のようにメンバー全員がひとつの目標に向かってひたむきに進むことが必要です。ベストを尽くせば、目の前の壁はいつか必ず崩壊し、前途が開けてきます。

■ おわりに

ターゲットに対して状態遷移表が適切である場合、ZIPC は大きな力を発揮します。

状態遷移表にしたら、仕様設計に多大な工数がかかります。引き換えに実装とテストの工数が大幅に減ります。バカ正直に道の真ん中を行けば、バグ撲滅ができます。

状態遷移表を実用化したZIPCを、使っていない人にはなかなか納得してもらえません。未来を見ているチャレンジャーは、他人に理解されません。成功したときに周りが気づくのです。やってみない人には、これが正しい道のひとつだと判断できません。

それとも「わかってるよ」と思っているんでしょうか。そして「でも・・・」と、使ってもいないのに、できない理由を考えているんでしょうか。

「将来の組み込みソフト開発はこうしたい」と夢を育てているあなた、理想の姿が見えるなら、我が道をゆくエンジニアを一緒にやりませんか。

（ねぎし まさと）