ZIPC適用によるソフトウェア開発プロセス改善

～半導体製造装置（ステッパー）自動化の制御ソフト開発における事例報告～

河村　紀夫

§はじめに　～理論は何度も聞いた！きちんと実行できるかが問題だ！！

ソフトウェアエンジニアリングの本を読みますと、品質や効率を上げたいなら開発の上流工程、つまり、要求分析や設計に力をいれて、洩れをなくすことが有効であるとほとんどの理論が説いています。しかし現実には、開発現場は理論に忠実には動いておりません。ソフトウェアの不具合の原因を整理しますと、プログラミングのミス（開発の下流工程のミス）に起因するものよりも、依然として、要求分析や機能設計の洩れがもたらした不具合の方が、発生数／影響（戻り工数）ともに無視できないものとなっています。そして、それらの分析設計の洩れは、納期圧力のために分析設計作業を一部分はしょって」しまったことに起因するものが、ほとんどです。つまり、理論はわかっていても、実行ができていない、のです。では、どうすれば問題を解決できるのでしょうか。

弊社のプロジェクトの1つにおいて、ZIPCを適用して経験したことをご紹介しながらソフトウェア開発プロセス改善を進めていく上でのヒントを探りたいと思います。

§設計が充分にできない事情

ソフトの品質や開発効率において、誰もその重要性を認める要求分析や設計がなぜ、現実的には充分にできていないのでしょうか。　技術者の能力の問題だと、簡単に片付けられるでしょうか。私は主に以下の2点が原因だと考えています。

1. 設計が、技術者の頭のなかでされるため、計量化／ビジュアル化が難しい。そのため、ある時点での分析／設計が充分なのか、まだカバーされていない部分が相当あるのか、良く見えない。従って「必要で充分な設計作業」に要する工数が数値化しにくい。

2. 納期に対する市場の要求が一層きびしくなるにつれて、開発期間の短縮化が進むと、「計画上、見えやすい」プログラミングやテストの期間が、先に確保される傾向があるため、「作業ボリュームをはっきりと示しにくい」要求分析や設計に充てられるべき期間が、主に削られることになる。

かくして、プログラミング開始予定時期が来ると、管理者と担当者の間で以下のような会話がされることになります。

「設計はどれだけ進んだのか？」
「おおどころは出来たのですが、まだ細部の詰めが残っていて...」
「それじゃ、出来た所からでもプログラミングに入ってもらったらどうだい？」
「.....」（このままプログラミングに入ると修正手戻りや洩れが心配でも、設計未完が何%あります！と明確に言えないし..）

この判断に苦慮されているプロジェクトリーダは多いと思います。多くの優秀なリーダの方は、ご自分が担当者として設計をしたときの経験と感覚で、プログラミング工程に進んで良いか、それとも、設計をもっと詰めるべきか、を決めておられます。　恥をさらすようですが、弊社も過去に手がけた開発プロジェクトに於いてこの判断の誤りのために、痛い目にあったことがあります。

設計が不十分であったとしても、プログラミングが終われば、見かけ上ソフトはできあがりますので、結合テストやマシン上のテストに入れます。しかしながら設計の洩れによる不具合の発見にともなう、解析作業や修正作業が発生するのは、当然です。その影響が大きなものであることは、繰り返すまでもありません。図1の例では、見かけ上のプログラム完成から合格水準の品質となるまで、実に5ヵ月も要してしまいました。

§ZIPCを適用して変わったこと

前節で例に挙げたプロジェクトAの後、弊社はほぼ4倍（ソース行数ベース）のプロジェクトBを経験いたしました。この規模でプロジェクトAのような事態が発生することを避けるため、何らかの形で設計の可視性を高めて、品質と開発効率を共に向上させることが、求められておりました。そのような背景から弊社はZIPCの適用を決めました。期待した効果は、以下の3点でした。

1. 装置のふるまいを規定するさまざまな条件を状態とイベントの組み合わせで洗い出し、状態遷移表の形で表現することによって、設計のボリュームを明確にする。
2. デザインレビュー時ならびに、設計からプログラミングへの移行段階での、コミュニケーションエラーを防止する。
3. 設計ドキュメントとして状態遷移表を使って、デバッグ工程での不具合が検出された時の、該当箇所の解析を迅速化する。

結果として、プロジェクトBの開発工程の推移は図2のようになりました。

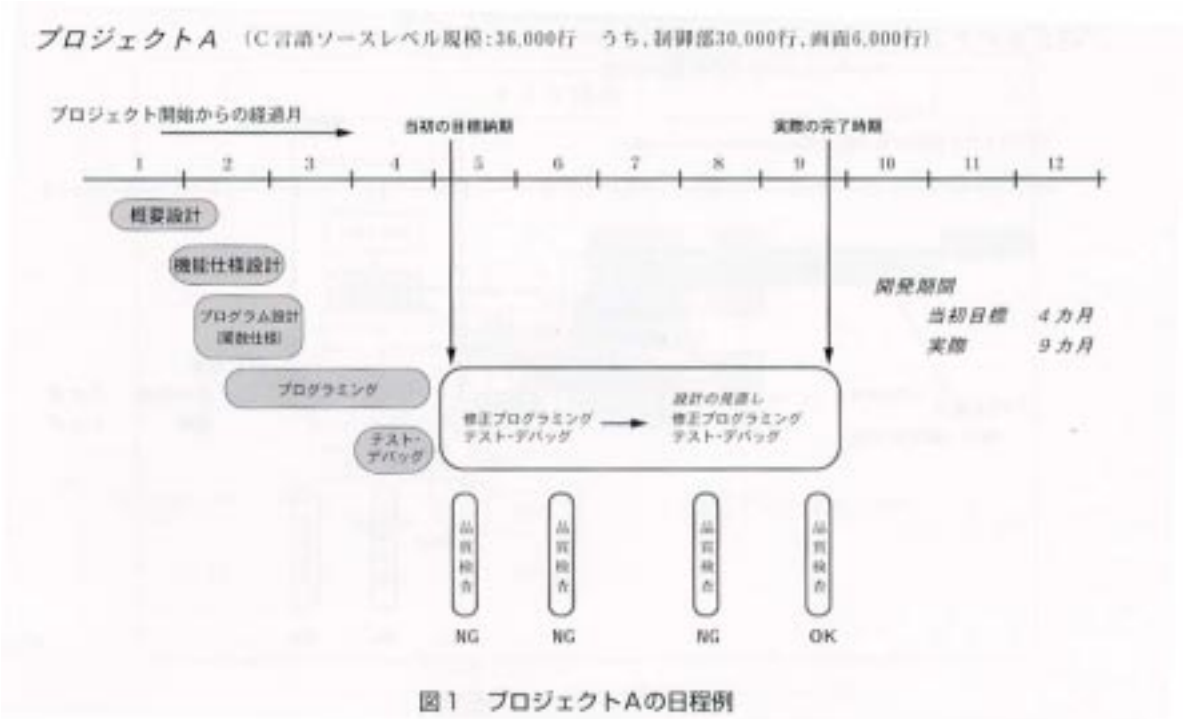

図1　プロジェクトAの日程例

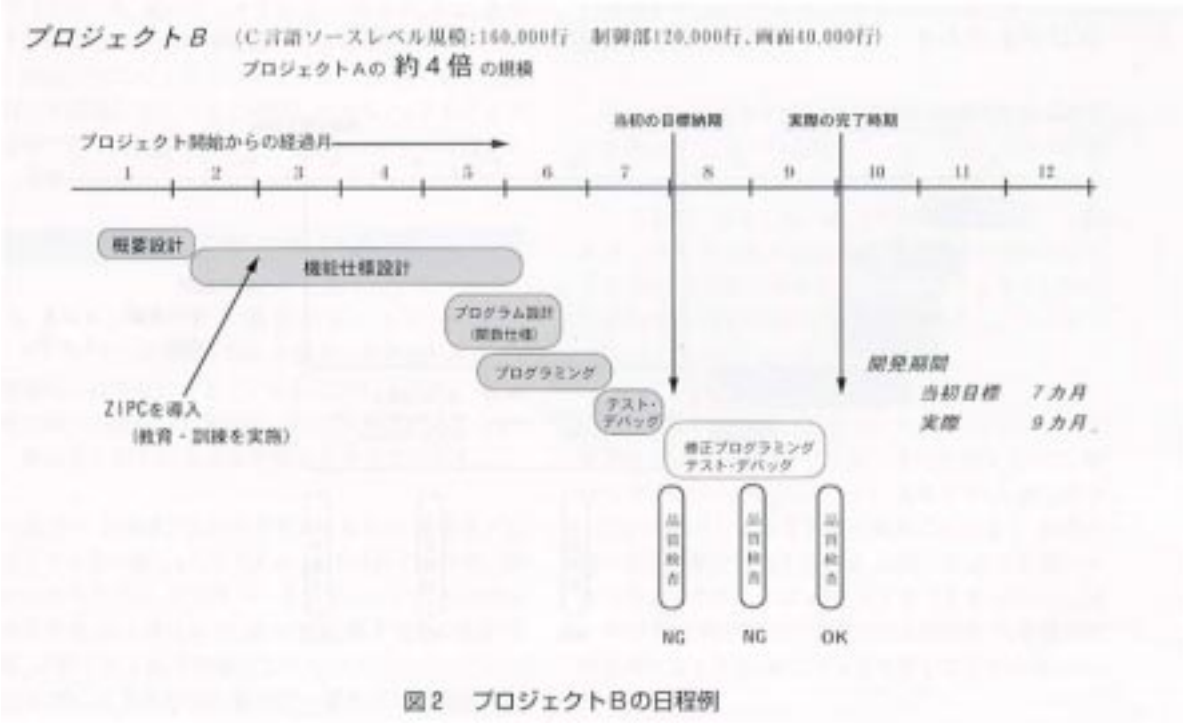

図2　プロジェクトBの日程例

残念ながら修正手直し期間がゼロという訳にはいきませんでしたが、それでも2ヵ月で合格水準にもっていくことが出来ましたし、プロジェクトAもほぼ同程度の開発期間で、プロジェクトAを大幅に上回る規模の開発を消化できました。

図1と図2を比較すると、開発期間に占める設計工程の比率に違いがあります。そこだけを見るとなにもZIPCを導入しなくたって、設計に比重をかければ良い結果が得られる、ということを示しているだけだ」と言う方もいるでしょう。おっしゃる通りです。ですが、「理論はわかっていても、実行ができていない」状態を解決したかった点を、再度思い出していただきたいと思います。

事実プロジェクトBの設計作業が長期化するにつれ、納期圧力が現場にかかってきました。プログラミングにどうして入れないのだ？いつまで設計するのだ？それだけ設計に時間をかけて下流工程で本当に挽回できるのか？のたぐいです。設計作業を簡略化してでも、次工程に早く入れるようにせよという指示がでたかも知れませんところが、プロジェクトBでは、そのようになりませんでした。その理由は、ZIPCの特徴と関係があります。

ZIPCの一番のインパクトは、状態遷移表形式であるために空欄のままでおくことを許さない点です。つまり、設計を中途半端のままやめられない、のです。ある意味では「窮屈さ」を感じさせますが、別の意味では「必要で充分な設計作業を守るための「頑固さ」として、開発工程の遵守に寄与するものでありました。弊社の例では、半導体製造装置の自動化システムにおける、いくつかのタスク開発にZIPCを適用いたしました。（図3、図4を参照）

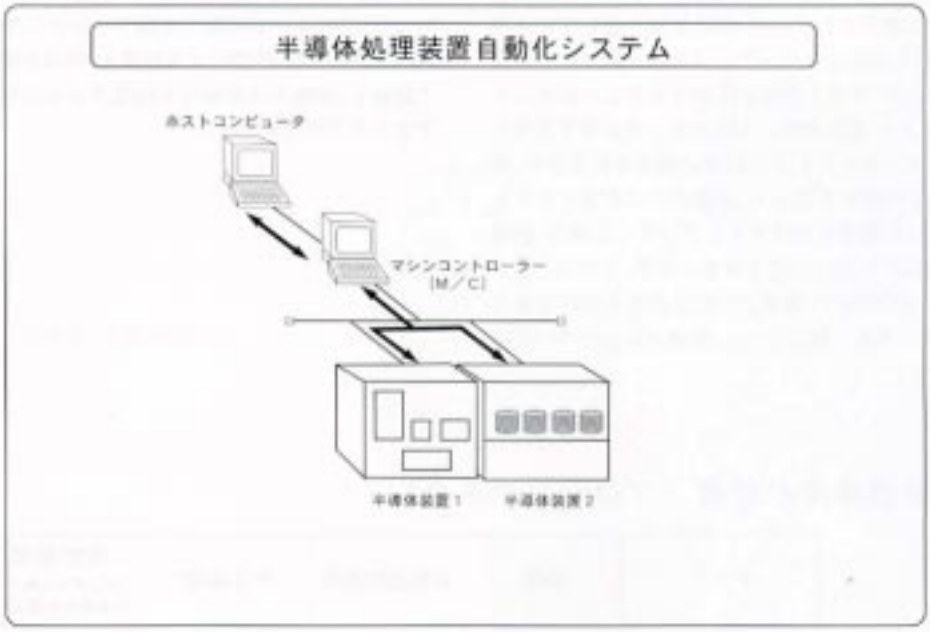

図3　半導体処理装置自動化システム

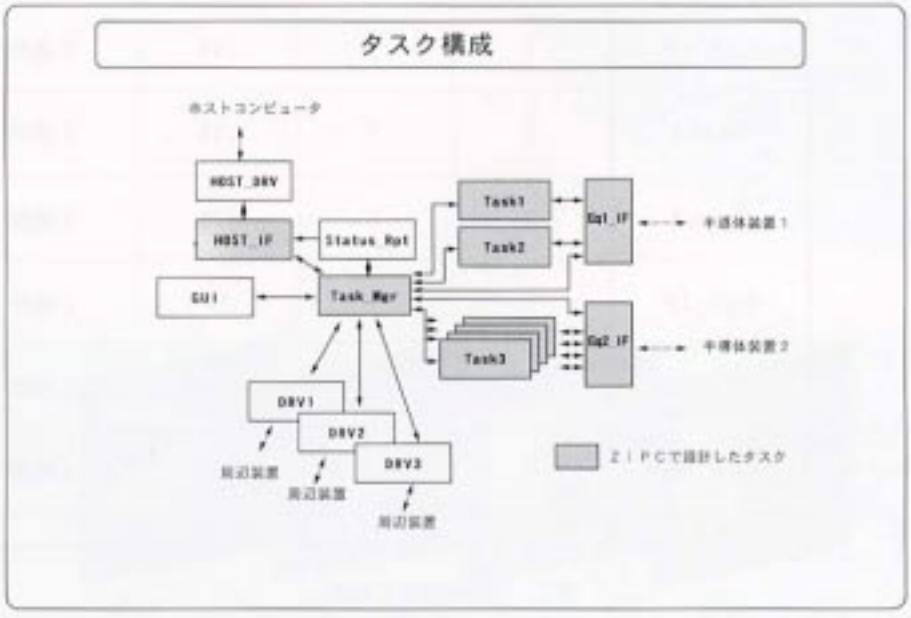

図4　タスク構成

作成した状態遷移表の規模は、遷移表の数で97、セル数で約4800でした。（表1を参照）

設計ボリュームの可視性という点では、従来設計方式に比べ、格段の改善であり、デザインレビューの進捗、セル総数のうちどれだけ終わったのかを報告することによって明確になりました。この可視性と、設計やレビューを完了しない限り、プログラミングに進めない、というZIPCの頑固さがプロジェクトBにおいて、設計作業に比重をかけることを実現した、といっても過言ではないと思います。

§設計ドキュメントとしてのZIPC

前節でご報告したZIPCに期待した効果の2番目と3番目にあたります が、「デザインレビュー時やプログラミング移行時のコミュニケーションエラー防止」「デバッグ時の解析作業の迅速化に関して、ZIPCは充分な効果をあげたと評価しています。いくら標準化しても、設計ドキュメントの詳しさのレベルが、担当者（個性）に依存してしまうことは、一度は経験されたことと思います。センスの問題だ、と言ってしまうとおしまいですが、デザインレビューをする時に、うまく整理された形の資料を使って行なった時と、そうでない時とは、レビュー効率／レビュー品質に差が出ます。

組み込み型ソフトは、さまざまな条件の組み合わせの処理を検証していくわけですから特に異常処理の部分について、条件を明確に特定しながらレビューしていくことがポイントになります。条件の組合せが膨大になりますと、検討／議論の途中でお互いに違うケースを検討していたりすることが起きやすくなるので、レビューの参加者はかなり神経を使うことになります。状態遷移表を使ってデザインレビューすると、セルを指すことで条件の特定が迅速にでき、コミュニケーションエラーを防ぐことができます。

ZIPCはジェネレーション機能があるので、設計～プログラミングの意思疎通のギャップは最小限となります。 もちろんジェネレーションの後で、各関数内のプログラミングをしなければならないので、その部分に関して設計者とプログラマとに誤解があると、それが不具合の原因になる可能性がありますが、少なくとも、制御の骨格部分は、確実に設計通りに仕上がる」という安心感があります。

この安心感は、デバッグ時の解析作業においても効果を発揮します。不具合発見時に、状態遷移表のどこを通ったケースで発生したのかを知れば、どの関数を修正する必要があるのか一目で判るからです。

設計ドキュメントと実際のプログラムとが一致している事がとても大切である点もわかってはいるが、実行できていない」ことであって、ZIPCによって実現できたことを評価しています。

状態遷移表の規模（プロジェクトB）

タスク	階層	状態遷移表数	セル総数	設計期間（プログラムモードの作成期間を除く）
HOST_IF	1	1	96	2週間
Eq1_IF	1	1	150	2週間
Task1	2	7	350	3週間
Task2	3	8	670	3週間
Eq2_IF	1	1	96	2週間
Task3	3	16	614	3週間
Task_Mgr	5	63	2823	6週間
合計		97	4799	

表1　状態遷移表の規模

§今後、ZIPCをどう活用していくか

弊社は、ZIPC新バージョン（ver.4.0w）を未だ、実際のプロジェクトに適用を試みていません。いくつかの新機能については、効果が期待できそうですので、試してみる価値はあると思います。なかでも注目しているのは、テスト自動検証機能です。理由は、以下のような状況があるからです。

新製品開発の時は、新規開発部分をテストする期間も日程上の配慮がされますが、製品を送り出してから後に、新機能を追加したバージョンアップ用ソフトを開発する時には、テスト期間が圧縮されることがあります。もちろん、追加機能に対応するために手を入れた部分のテストをするだけの期間は確保されますが、修正していない部分も含めた、全部のパスを通すテストをやるだけの期間を確保することは多くの場合、納期圧力との関係で、難しいことが多いです。このことがデグレードバグの形で、品質上の弱点となる恐れがあるわけです。これも、「修正したら関連する全てのパスを通して確認する」という基本理論がわかってはいるが、実行できていない」ひとつの例です。

ソフトウェア開発におけるプロセス成熟度モデルをご存知の方も多いと思いますが、高度な段階を実現する前に、やれば効果があるとわかっていることを確実に実行する」ことで顕著な改善が図られる例が、現場ではまだまだあるように感じます。今回ご報告した事例のように、ZIPCは開発支援ツールとしての効果と共に、開発工程の改善による効果を現場が実例をもって認識し、組織の体質強化を推進するものとして活用できると考えています。

河村　紀夫（かわむらのりお）
キヤノン株式会社 半導体機器開発センター 半導体機器ソフトウェア開発室
kawamura@cks.canon.co.jp