開発ドキュメントの期待される姿　2011．06．17　PMファクトリー

索引

１．ドキュメントの役割について

開発ドキュメントをどう作るべきかって話の前に、本来開発ドキュメントは何のために、またどんな役割なのか、その本質について考えてみよう。

１．１　抽象的な見えない世界と具象的な見える世界について

人間の発明品は頭の中でひらめいたアイデアやイメージを実際に使えるモノとして具現化したものだ。頭の中のイメージは見えない世界のものだ。一方、作られた発明品は実際に誰の目にも見え、触れて、動かすことのできるものだ。

見えない世界とは頭の中の抽象的なイメージの世界のことだ。例えば、最初に自動車を作ろうとしたときに頭の中に浮かんだイメージが抽象的なイメージなのだ。

見える世界とは、現実に人間の五感で確認でき物理的な作用が行われる具象的なモノの世界のことだ。例えば、目の前にある実際の自動車そのものが見える世界のモノなのだ。

１．２ ドキュメントは見えない世界と見える世界の仲立ちをするものだ

開発ドキュメントは見えない世界と見える世界の仲立ちをするものだ。頭の中のイメージだけではモノをつくることは難しい。一人でできるような簡単なものならばイメージだけでもモノは作れるかも知れないが、複数の人間で複雑なものをつくる場合は頭の中のイメージだけではモノは作れないだろう。
ドキュメントは頭の中のイメージを、擬似的に見える世界に近いものとして表現したものなのだ。ドキュメントはまだ抽象的なものなのだ。

１．３ ドキュメントには限界がある

イメージをドキュメントにおとす過程で多くのあいまいさは排除され正確な情報に転換されドキュメントに表現される。しかし人間のイメージそのものが非常にあいまいな状態であり、また発明品自体はまだ誰も見たことがないのだから誰も最終的な姿がどんなものなのか正確に表現することは不可能なことなのだ。
だからドキュメントは多くのあいまいさを含んでいることは必然的なことなのだ。モノを作る前に用意される開発ドキュメントが成果物のモノを完全に表すことは不可能なことなのだ。
ドキュメントの限界をカバーするものがフェイス・トゥ・フェイスのコミュニケーションだ。ドキュメントという文書およびフェイス・トゥ・フェイスの会話による二つのコミュニケーションが意思疎通の基本原則なのだ。

1．4　どうしたら伝わるか

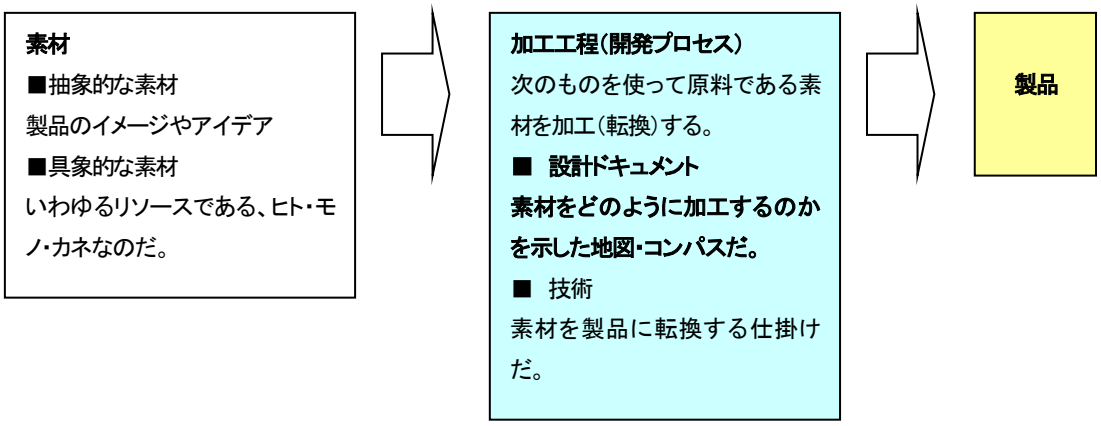
どうしたら伝わるかについて的確に表現された一文を紹介しよう。

“確実にキャッチされる球を”
「キャッチボールで大事なのは投げることではなく、受けてもらうこと。話すことも書くこともそれと一緒で、情報量を増やしても、伝わらなければ意味がない。いくら豪速球を投げてもダメなんです。「伝える」ではなく「伝わる」に意識を置かねばならない。」［山川静夫　エッセイスト・元NHKアナウンサー］

1．5　開発における設計ドキュメントの役割

モノ作りって結局、何らかの素材を加工して具体的な製品を作りだすことだろう。素材ってどんなものがあるのだろうか。すぐイメージされるのは実際に手に触れられるようなものをイメージするだろうが、それだけじゃないだろう。手に触れられないもの、例えば製品のイメージや発明のアイデアなども素材だと思う。これらの素材を加工するのに必要なのが技術であり、その技術をどのように使うのかを示したものが設計ドキュメントだろう。この加工の一連の作業のことをみんなは開発プロセスと呼んでいるのだ。

モノ作りの全貌を絵に描いてみると次のようになるだろう。



１．６ 開発ドキュメントの要件

開発ドキュメントの代表格である設計書について必要な要件は次のようだろう。

1. **正しいこと**
   このドキュメントを見て正しく動作するプログラムが作れること。正しいとは、顧客要件を過不足なく全て満たしているということだ。
2. **矛盾点がないこと**
   要求仕様書と矛盾した内容がないこと。要求仕様書自体が矛盾点を含んでいた場合は要求仕様書に遡って顧客要求に合致する姿に修正すること。
3. **正確であること**
   書かれている内容が正確であること。すなわち定量的であること。定性的な表現、例えば”非常に”とか”かなり”などの副詞的あるいは形容詞は技術的ドキュメントでの使用は避けたい。
4. **分かりやすいこと**
   分かりやすく記述されていること。複雑なことが簡潔に表現されていること。
5. **モレがないこと**
   行間を読む必要がないこと。
6. **誰が読んでも同じ解釈がなされること**
   誤解の余地がなく論理的・科学的に記述されていること。

開発における他のドキュメントについても同じ原則が適用できるだろう。

2.　開発における開発ドキュメントの現状

2．1　開発者たちの生の声

☆　誰のために書かれたものか分からないドキュメントが多い。作成者以外が見ても分かる内容・レベルになっていない。
☆　納期が間に合わなくなったらテスト作業やドキュメントを省略してしまう。
☆　内容が更新されていない。
☆　システムの全体を表した資料がない。
☆　分かりにくいドキュメントばかりしかない。
☆　機能を実現するために必要な情報が設計書に記述されていない。
☆　客先要件の実現方法が明確に書かれていない。
☆　開発内容に対するコンセプトや背景や経緯についての資料がない。
☆　システムの全体を表した資料がない。
☆　品質は上流工程（要件定義書・設計書）で決定する。テストだけでは品質は改善しない。
☆　客先要件や仕様内容の記述が乏しくロジック記述に偏っている。運用テストに使えない。
☆　障害解析に使用できるレベルの内容になっていない。
☆　障害報告書で技術的側面から図表等を使用した報告ができていない。
☆　モジュール間の連携部分の説明が貧弱。
☆　既存ソフトの流用の可否を判断できる資料になっていない。
☆　整理・管理されていないため調査・検索ができない。

これが大方の現実なのだろう。こんなんでいいのか！

2．2　なんでそうなるの

☆　前任者がドキュメントを更新していなかったから、自分もできなかった。
☆　協力会社に設計以降の工程を丸投げする方が安く・納期短縮もできたから。
☆　バージョンアップ開発時にはソースベースで調査・改修でも何とかなった。
☆　発注元からドキュメントを要求されることもなくドキュメント作成工数が削減できた。
☆　発注元が協力会社に丸投げする状況が続き、発注元の担当者が自ら設計する技術がなくなってきた。
☆　時間に余裕がなく、とりあえず自分が分かるレベルの内容記述だけになった。客先・SEや他の開発者・協力会社が理解できる内容にはなっていない。
☆　時間も人もいなくて、限定した内容だけの記述しか作成できなかった。更新が必要な資料にも手をつけられなかった。

何でそうなるの？

2．3 開発ドキュメントはいつ作るの？

大切なのはプログラムを作ることだけですか？ ドキュメントは一応作っておけばいいものですか？
本来、要件定義書や設計書は何のためにあるのだろうか。
開発ドキュメントは、開発者が正しく設計・開発できるように客先要求を開発の言葉に変換したものだ。何をどういう風に作れば良いかを可視化したものであったはずだ。決してプログラムを作りながらとか作った後に作るものではないはずだ。
開発ドキュメントは目標にたどりつくための地図のようなものだろう。先に地図がなければ目的地には到達できないだろう。どこの世界に目的地らしい所についた後で、後付け的に地図をつくる人がいるだろうか。そのような人やチームがあれば道に迷うか遭難するかのどちらかだろう。

今やオフショア開発が常態化する中でプログラム作成能力しか持っていない日本のプログラマーの役割はなくなってしまったも同然なのだ。日本で生き残ることができるソフトウェア技術者は優れたプロジェクトマネジャー、ドキュメント作成能力のある設計者ないしは三倍の生産性を持つスーパープログラマーくらいしかないだろう。

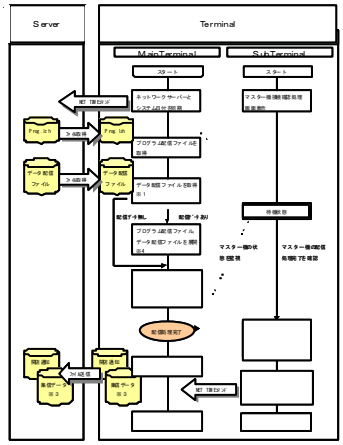
2．4 開発業務における主なドキュメント

開発における主要ドキュメントは次のようだろう。

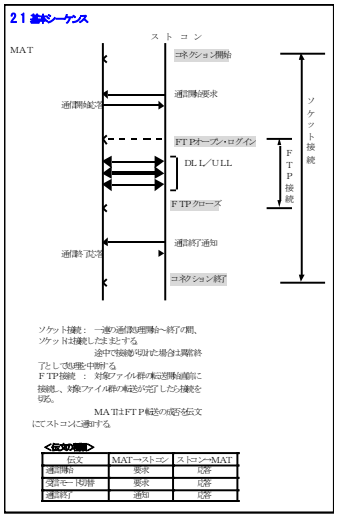
- □ 要求仕様書（要件定義書）
- □ 設計書
  設計書には下記のようなさまざまなドキュメントがあるだろう。
  概要設計書、詳細設計書、業務運用フロー、データフロー 、システム論理構成図、
  プロセスフロー、ソフト構造図、修正影響度表、インターフェース仕様書、など。
- □ 手順書・ガイドライン
  見積りガイドライン、プロセスガイドライン、設計手順書、コーディング規約書、評価手順書、
  構成管理手順書、など 。
- □ 計画書
  プロジェクト計画書、テスト計画書、など。
- □ 見積書
- □ 企画提案書
- □ 開発管理表
  予算管理表、進捗管理表、労務管理表、機材管理表、成果物管理表、障害管理表、など。
- □ ソースコード（コンピュータ言語記述書）
- □ 技術メモなど。

2.5 設計図の例

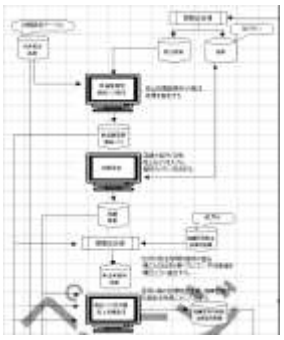
機能仕様書



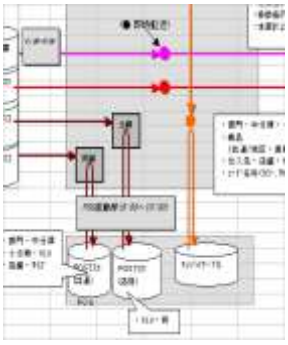
インターフェース仕様書



業務運用フロー



データフロー

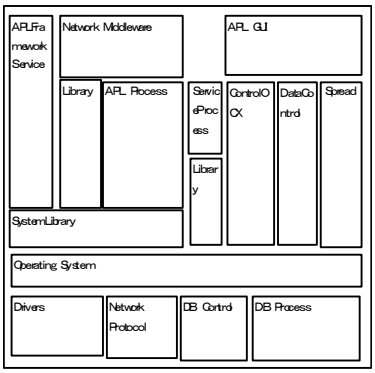


プロセスフロー

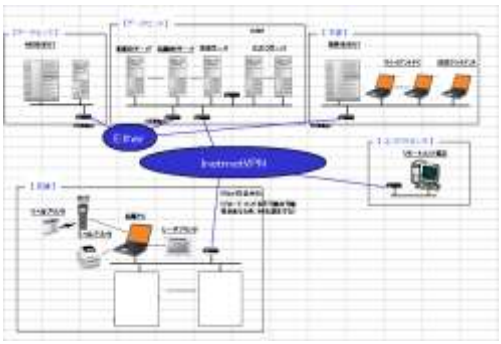
修正影響度表

構成管理手順書

ソフトウエア構造図



システム論理構成図



2．6 「設計図がない」状況からの脱却

日経エレクトロニクス誌における文章を引用する。

『 いきなりコーディングを始めていませんか。設計図も作らずに

多くのエンジニアリング分野では、モノを作り始める前に、まずは設計図があります。実は現在、ソフトウエア開発において「設計図」らしきものはないに等しいと思われます。あるのは100万行を超える膨大なソースコードの束だけでしょう。

ソフトウエアはコンピュータ誕生の当初から、コンピュータに対する命令が文字で表現されてきました。いわゆる機械言語と呼ばれるもので、以来、機械語、アセンブリ言語、C言語、Javaと高度化しましたが、常に文字で書かれた「文章」でした。

難解なソースコードを読めるのが一人前のソフトウエア技術者であるとの認識が業界の常識でした。 しかし、時代は変わりました。組込みソフトウエアの規模は2000年ごろを境に100万行を超えました。 携帯電話に至っては1000万行に達する勢いです。既に一人の技術者が全貌を把握できる規模ではなくなっています。

今必要なのは、開発にかかわる技術者すべてが全体や細部の構造を把握できるようにするための、ソフトウエアの設計図です。 文章だけではなく図で表すことで、技術者の理解度は段違いに高くなります。

今後、ソフトウエアの開発は、プログラミングではなく、設計図（モデル）を作ることを意味するようになります。パソコンの前で、徹夜でプログラミングのためにキーボードをたたき続けることだけが、ソフトウエア技術者の仕事ではないのです。

こうした設計図中心のソフトウエア開発を実現するための手法が「モデル・ベース開発」です。 』

［出典:「設計図がない 脱プログラミング至上主義」 日経ｴﾚｸﾄﾛﾆｸｽ 2006.9.11 進藤智則］

事実、1999年当時のPOSレジスターにおいてもすでに100万行を越えていた。もうソースコードからソフトウエアの構造を読み解くことはほとんど不可能だろう。

ソフトウエアが大規模化・複雑化していく状況において設計図も構造図もない状態では多数の技術者・評価担当者・他のプロジェクトの開発者等にソフトウエアの構造・機能を説明し理解してもらう事は全く不可能なのだ。

コスト圧力が一層の厳しさを増しオフショア開発が急激に拡大する中で、プログラミング作業はすべて中国を筆頭にした海外に渡り、日本に残れるソフトウエア開発組織における価値のある仕事の一つは要件定義書および設計図の作成ノウハウに基づいたプロジェクトマネジメントだけになったのかも知れない。

2．7 ビジネスピンチ？それともチャンス？

設計図・構造図を書けない・書かないためどれだけの損失が発生しているのか考えてみよう。
有効なドキュメントがないために発生する損失として、不要な開発費用、やり直し費用、バグつぶし費用、再評価費用、市場対応費用、手待ち時間ロス、対策の横展開未実施によるロス、部材の流用・共用不可能によるロス、技術者のレベル向上阻害による長期にわたる人的生産性のロスなど数え上げたら切りがない。目には見えないかも知れないが数十パーセントものロスになっているのではないだろうか。

日本におけるソフトウエア開発企業は今、要件定義・設計図・構造図が書ける技術者やプロジェクトマネジャーの育成に投資を惜しむべきではないだろう。

日本で生き残る道は他にはないと思う。オフショアによる受注金額・仕事量の大幅減、社内におけるロスの撲滅、新たな付加価値の高い仕事の獲得の三つを同時に解決する方法は要件定義書・設計図・構造図が書ける技術者やプロジェクトマネジャーをどれくらい揃えられるかにかかっていると言っても過言ではないだろう。

日本の市場ではプロジェクトを統括できるプロジェクトマネジャーを始めとして、お客様の言葉をシステムの言葉に変換できる要件開発技術者、要件定義の言葉をさらに開発者が理解できる言葉に変換できるシステムの設計図・構造図が書ける構造設計開発者の需要は限りなく存在するだろう。

3.　なぜ満足のいく開発ドキュメントが作れないのか

何故作れないかって言われてみても、チャントやらないからできないだけだとか、つくるヒマがないからできないのだ、なんて反応しか返ってこないようだ。作りたくなければ地図もコンパスもない困難な開発と爆発の評価工程と炎上の市場展開を覚悟するしかないのだ。

ドキュメントの作成は抽象的なものをある程度目に見えるように紙に書く知的で難しい作業だということを再認識しておいた方がいいのだ。ある決まった素材を与えられた場合に作成されるドキュメントはどこの誰が作っても同じようなレベルのドキュメントが出てくるわけでもないだろう。ドキュメントの質を上げたければ、その質を左右するものについて知っておいた方がいいだろう。

ドキュメントの質は大きくいって**開発者の能力、開発手法、開発組織体制、開発組織文化**の四つに依存していると思う。

3．1　ドキュメントは開発者の能力に依存する

作成されるドキュメントは、その作成者である開発者の能力以上のものにはなり得ない。必要な能力には、技術的能力、言語能力、抽象的なものをより具象化できる能力などがあるだろう。いいドキュメントを作りたければ人材育成を怠らないことだ。

3．2　ドキュメントは開発手法に依存する

作成されるドキュメントは、その開発で使用される開発手法に大きく影響されるだろう。ここでいう開発手法には、ソフトウェアの構造構築の方法や、ウォーターフォールやアジャイルなどの手法の他にWeb・OS・ミドルウェア等の技術インフラも含んでいる。
ベースとなるソフトウェアの構造体やインフラが複雑で分かりにくいものだったら、ドキュメントも複雑で分かりにくいものになってしまうだろう。
分かりやすさ・正確さ・速さに関するキーワードは、Simple（簡単）・Short（短い）・Structured（構造化されている）・Compact（まとまりの良い）・Light（軽い）・Fast（早い）、などだろう。
いいドキュメントを作りたければ、いい開発手法やすっきりした構造体を採用し、上記3つのSとCLFのキーワードを実践することだ。
個人的に注目しているものにアジャイル（スクラム、XP等）、XDDPなどがある。

3．3　ドキュメントは開発組織体制に依存する

会社組織の中においてはそれぞれの役割の部署において担当のドキュメントが作成されるだろう。ある程度以上の組織規模を持つ会社においては、開発組織はその役割に応じていくつかの組織に分割されているだろう。例えば企画部、要件開発部、設計部、評価部、品質保証部などがあるだろう。組織の数が増えれば増えるほど組織間の関係は複雑になり、連携不足やコミュニケーション不足が発生するだろう。そのような組織環境の中でそれぞれに作成されたドキュメントの統一性・整合性・正確性・分かりやすさを保つことは非常に困難だろう。
いいドキュメントを作りたければ、組織体制はSimple（簡単）、Compact（まとまりの良い）、Structured（構造化されている）、Fast（早い）などの体制にしておくことだ。

３．４　ドキュメントは開発組織の文化に依存する

組織の文化とは、その組織の習慣になっている開発スタイルとかやり方のことだ。良い習慣や良い開発スタイルを持っている組織からはレベルの高いドキュメントが出てくる確率は高いだろう。悪い週間や悪い開発スタイルの組織からはレベルの低いドキュメントしか出てこないだろう。

悪い組織文化のいくつかの例をあげてみよう。

① 他者依存的姿勢による、協力会社への丸投げ、部下への丸投げ体質
② 情緒的人間関係
③ 非科学的な情緒的思考
④ 情報軽視、データ軽視
⑤ 認識力の弱さ
⑥ コミュニケーション機能不全
⑦ 戦略の欠如（問題に対する取り組み姿勢の弱さ）

組織の構成員である開発者たちは間違いなくこれらの悪影響を受け、彼らの作成するドキュメントも同様の影響を受けたものになるだろう。自分では作らない、情緒的な表現、データ軽視、機能重要性プライオリティの認識のない、情報に整合性のない、自分では解決する意思のないドキュメントしか作れなくなるのだ。

いいドキュメントを作りたければ、自律的・合理的・科学的・データ重視・認識力の強い・密なコミュニケーションが実行される・問題に対する取り組み姿勢のしっかりした、組織文化を作り上げることだ。

３．５ 開発ドキュメントを作るにあたっての基本的な考え方

以上をまとめると、いいドキュメントをつくるには次のことを実行するしかないだろう。

① 開発者について

開発者は、技術的能力、言語能力、抽象的なものをより具象化できる能力・技法などをマスターし、会社はそれを支援することだ。社内・社外にあるベストサンプルを収集し真似ることから始めてもいいのだ。

② 開発手法について

いい開発手法やすっきりした構造体を採用し、Simple（簡単）・Short（短い）・Structured（構造化されている）・Compact（まとまりの良い）・Light（軽い）・Fast（早い）を実践することだ。

従来のうまくいかない手法にいつまでもしがみついていないでアジャイル、XDDPなどからそのエッセンスを吸収し自分の仕事に生かそう。

参考になる書籍をいくつか紹介しておこう。

- アジャイルソフトウェア開発スクラム　ケン・シュエイバー＋マイク・ビードル著
- XPエクストリーム・プログラミング入門　ケント・ベック著
- 「派生開発」を成功させるプロセス改善の技術と極意　清水吉男著
- 要求を仕様化する技術　表現する技術　清水吉男著
- システムクリエイツHP　http://homepage3.nifty.com/koha_hp/
- 製品開発革新（第 9 章　ラグビー方式による新製品開発競争）　竹内弘高・野中郁次郎等共著
- 日経システムズ　20011年 4 月号　特集1　開発の常識　これからはファストプロセスだ

③ 開発組織体制

組織体制はSimple（簡単）、Compact（まとまりの良い）、Structured（構造化されている）、Fast（早い）などの体制にしておくことだ。

ある程度以上の職位にないとこの実践は難しいかも知れないが、小さいチームから、小さい課から実践を開始していくのだ。

④ 開発組織の文化

自律的な・合理的な・科学的な・データ重視の・認識力の強い・密なコミュニケーションが実行される・問題に対する取り組み姿勢のしっかりした、組織文化を作り上げることだ。

これが一番やっかいな問題だが、上記①、②、③の積み重ねがスピードと柔軟性をもつ開発組織文化を作り出すのだ。

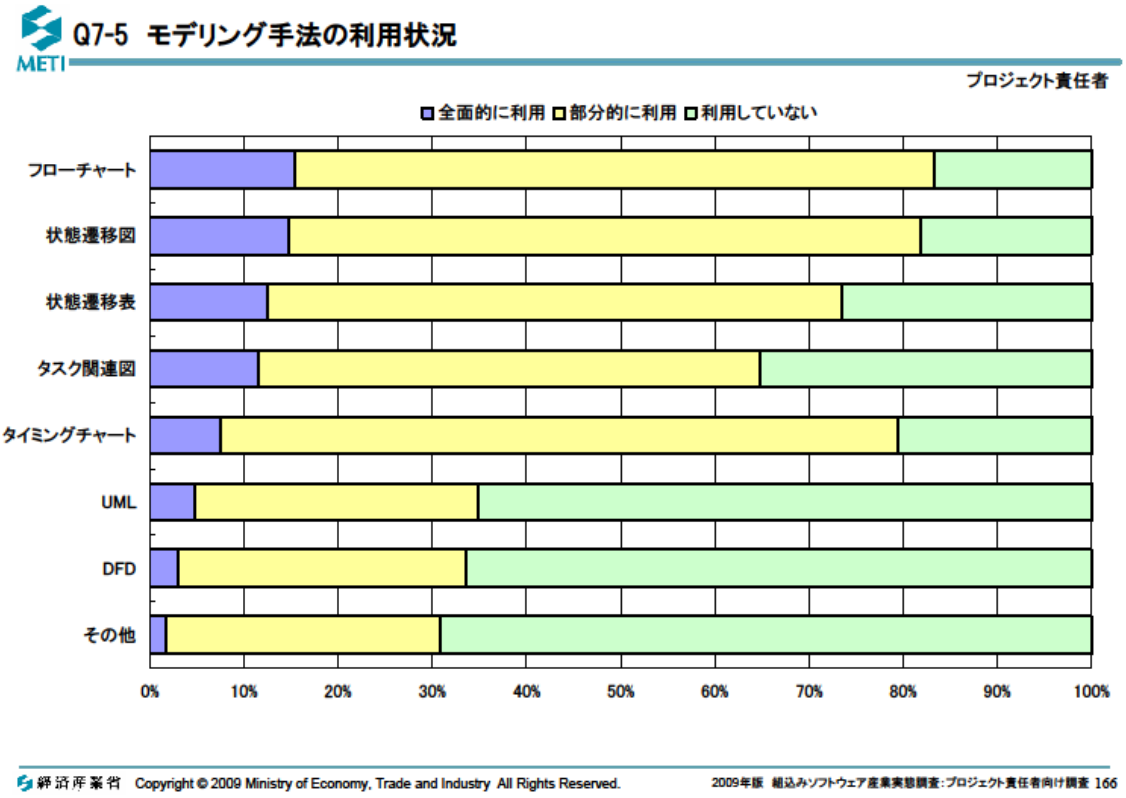
全部一挙に実現することは難しいだろう。まずは自分のできる範疇から実行していこう。

4.　モデルベース開発の進め

　これまでのソースコード中心の開発から設計図中心の開発へ移行する手法の一つとしてモデル・ベース開発がある。
　従来の手法ではユーザ・要件定義者・プロジェクトマネジャー・開発技術者・評価技術者間のコミュニケーションを行なうための全体を通しての共通言語は存在していない。 そのためシステムの巨大化・複雑化も相まってユーザの要求とおりの製品を実現することは極めて困難なのだ。
　モデル・ベース開発は上流工程の要件開発の品質向上に大きく貢献するだけではなく、開発に係わるユーザをはじめとする全関係者を結ぶ共通言語を提供しQCDの全てに貢献するものと思われる。

4．1　モデリング手法の利用状況

　下記は 2009 年 8 月に経済産業省情報処理推進機構より発表された日本の組込みソフトウエア産業におけるモデリング手法の利用状況のデータだ。



　モデリング手法の世界標準であるUMLのプロジェクトでの使用率は、全面的に使用が4．8%、一部使用が30%で合計34．8%とまだまだの状況だ。
　世界的には90～70%程度のようで、すでに常識になっているようだが日本はかなり出遅れているのだ。

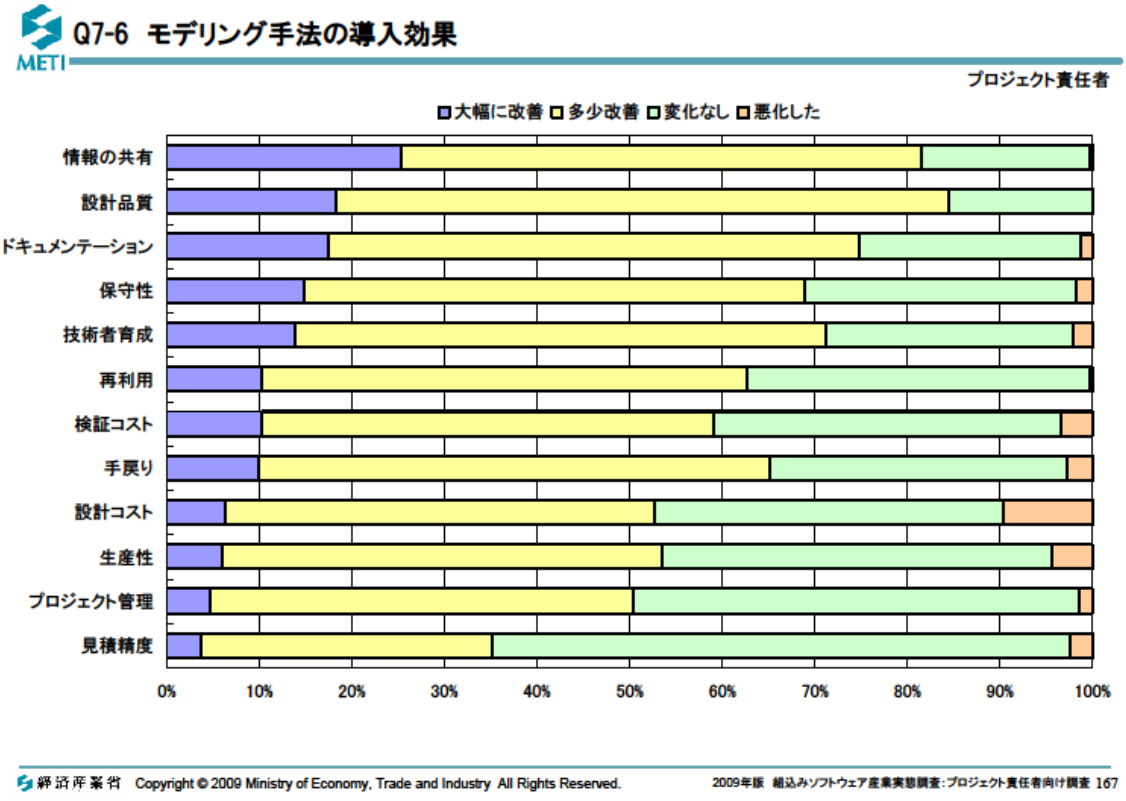
日本で普及しない言い訳
- 顧客も勉強してないから顧客との共通言語にならない。

・ モデル化・オブジェクト指向の概念の学習がむつかしい。
・ 設計工数が増える。

いつまでもそのようなことばかり言っていられるのだろうか？

4．2 モデリング手法の導入効果

下記は 2009 年 8 月に経済産業省情報処理推進機構より発表された日本の組込みソフトウエア産業におけるモデリング手法の導入効果のデータだ。



表で明らかなようにモデリング手法の効果が大きいようだ。
大幅に改善した項目は以下の通りとのことだ（括弧内は“多少改善した”数値を加えたもの）。

情報共有 25．3（81．6）%、設計品質 18．2（84．4）%、ドキュメント 17．5（74．8）%
保守性 14．9（68．9）%、技術者育成 13．9（71．2）%、再利用 10．3（62．6）%
検証コスト 10．3（59．1）%、手戻り 9．9（65．2）%、設計コスト 6．3（52．7）%
生産性 6．0（53．5） %、プロジェクト管理 4．7（50．4）%、見積精度 3．6（35．1）%
“多少改善した”を加えれば圧倒的に効果があったとのアンケート回答になっていた。

4．3　モデリング手法の導入

日本における過去の取組み状況から見て現実的な導入方法は、UMLだけにこだわることなく自社のソフトウエアの性質および技術レベルを見極めて、非UMLのモデルも取り入れた形で、開発の各工程に適切なモデル図を使用することがポイントだ。要は現状開発における曖昧さ・不透明さを排除し、お客様と開発が共通の認識をえられる方向性で進むことにあるだろう。

下記に主なモデルの表記法を示した。

	データ構造のモデル（静的モデル）	振る舞いのモデル（動的モデル）
UML表記	●クラス図 その他に、 〇コンポーネント図 〇配置図 〇オブジェクト図 〇パッケージ図	●ユースケース図 ●シーケンス図 ●アクティビティ図 ●コラボレーション図（相互作用概念図） ●ステートチャート図
非UML表記	●DFD（データフロー図） ●ER図 ●ディシジョンテーブル	●フローチャート ●CFD（制御フロー図）
非UML表記	●画面遷移仕様書 ●性能・消費電力等の非機能要件 ●その他、分野独特の表記法（制御ブロック線図など連続系のモデル表記法） ●形式仕様言語などテキスト言語によるモデル表記法	

4．4　そうは言っても

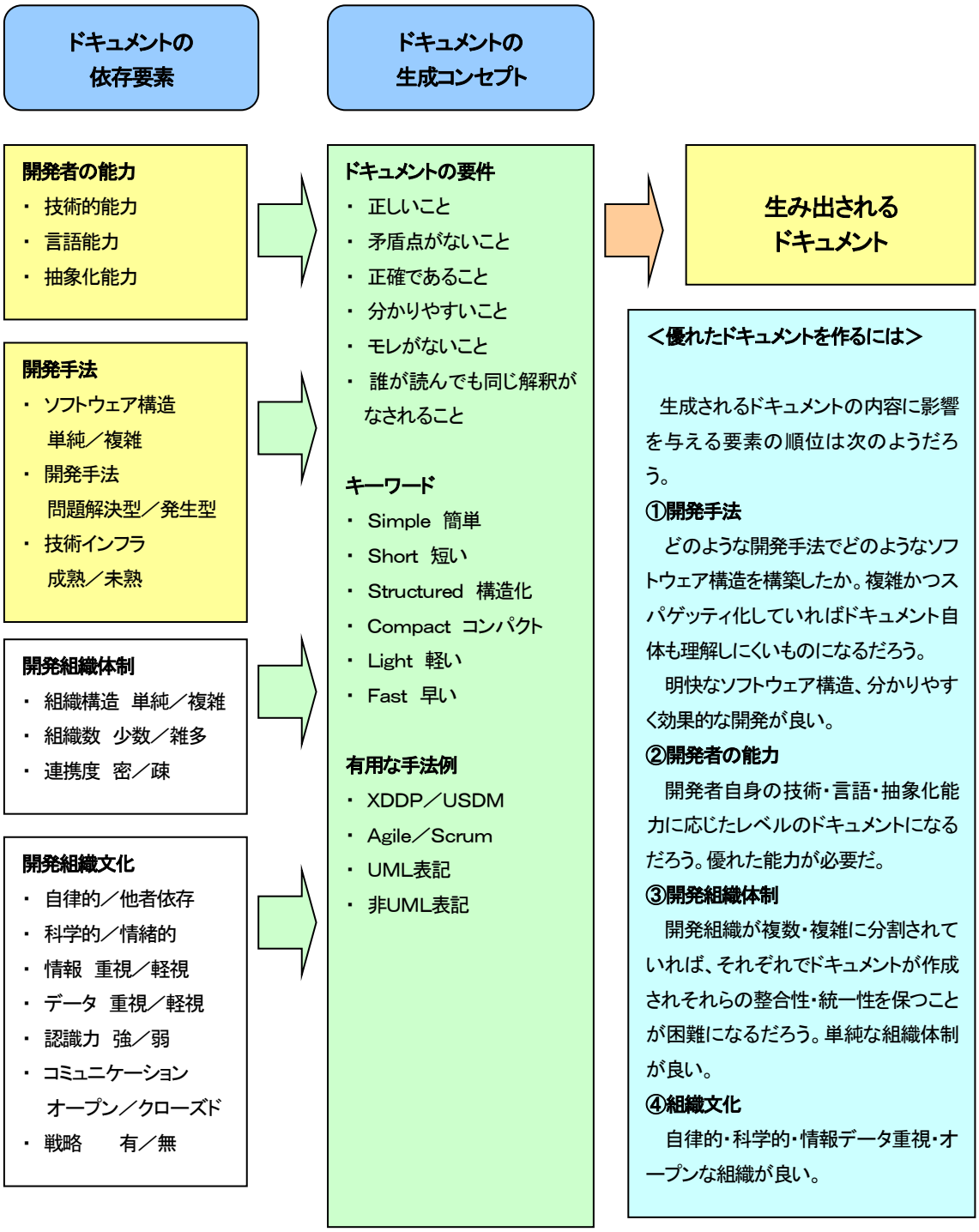
自然言語である日本語ですら満足に書けないのに抽象化の極みであるUMLなど、レベルの高くない開発組織においては使いこなすのは無理なのかも知れない。

UML表記にこだわらず自社のレベルに合わせて非UML系の優れた表記法を採用するのも一法かと思う。 また派生開発においてはXDDPの採用は非常に有効な方法の一つだろう。

［XDDP（参照　派生開発を成功させるプロセス改善の技術と極意、清水吉男著）］

5.　どんなドキュメントが何からどのように生み出されるのか

下図は開発において作成されるドキュメントのレベルがどういう要素に依存して生み出されるのかの因果関係を図式化したものだ。



[図. 開発ドキュメントの生成因果関係]