「要求仕様定義ガイドライン」

～UVC（User Vender Collaboration）研究プロジェクト報告書～

０．まえがき・JUAS　UVC 検討プロジェクト事業体制

１．このプロジェクトの目的と背景

２．要求仕様書に関わる問題とその解決方法

３．要求仕様書とは

４．要求仕様書作成に関わる作業

５．知識共有の方法

６．要求仕様の定義方法

７．ユーザ部門と IT 部門の役割

８．要求仕様を設計からプログラムまでトレースする方法

９．データベースの定義方法

10．画面／帳表のデザインとユーザビリティ

11．非機能要求の定義方法

12．USDM を使用した要求仕様書の有効性の確認

13．USDM 表記法の適用方法

14．残された課題

付１．参考文献

付 2．要求仕様書のプロトタイプ

付 3．要求仕様書のテンプレート

付 4．画面／帳票定義書の記載項目

付 5．「USDM 表記法による要求の仕様化について」

経済産業省　情報処理振興課

社団法人　日本情報システム・ユーザー協会（JUAS）

「要求仕様定義ガイドライン」

～UVC 研究プロジェクト報告書～

2007 年（平成 19 年）3 月 31 日

経済産業省　情報処理振興課

社団法人日本情報システム・ユーザー協会

目　次

０．まえがき・JUAS　UVC 検討プロジェクト事業体制 3
１．このプロジェクトの目的と背景 9
２．要求仕様書に関わる問題とその解決方法 18
３．要求仕様書とは 22
４．要求仕様書作成に関わる作業 34
５．知識共有の方法 45
６．要求仕様の定義方法 53
７．ユーザ部門と IT 部門の役割 63
８．要求仕様を設計からプログラムまでトレースする方法 70
９．データベースの定義方法 75
10. 画面／帳表のデザインとユーザビリティ 83
11. 非機能要求の定義方法 93
12. USDM を使用した要求仕様書の有効性の確認 117
13. USDM 表記法の適用方法 130
14. 残された課題 135

付１．参考文献 137
付 2. 要求仕様書のプロトタイプ 140
付 3. 要求仕様書のテンプレート 141
付 4. 画面／帳票定義書の記載項目 142
付 5. USDM 表記法による要求の仕様化について 143

0．まえがき・JUAS　UVC 検討プロジェクト事業体制

1. 前書き

システム開発についてユーザとベンダーの間には深い溝がある。

発注する側は、「こんなものが欲しい」と言って欲しいシステムができてくるのが理想ではあるが、それだけでは受注する側の SE には、「こんなもの」の意思、内容は伝わらない。ユーザ側（発注側）は明確な仕様情報をベンダー（開発を受注する側）に伝える必要がある。

日本情報システム・ユーザー協会（JUAS）ではユーザー満足度調査を実施しているが、ユーザ側が自分自身で要求仕様書を書いて開発してもらった場合と、仕様作成をまったくベンダー任せで開発した場合では、明らかに仕様を明示したプロジェクトの方が満足度は高い結果がでている。「当社は人不足で要求仕様書を自分でまとめられないが、よろしく頼む」と発注した場合の結果は、おおかた期待したものにはならない。

では、要求仕様はどのように書けばよいのか。SLCP-JCF98 に見積要求仕様書の中にはこのような項目を盛り込みなさいと 43 項目が記載されているが、その内容をどのように書けばよいのかは明示されていない。わずかにシステム化の依頼範囲、依頼内容、業務の詳細を書けと載っているだけである。

ところが、システム開発プロジェクトで失敗すると、ベンダーの皆様は「要求仕様書の出来が悪く後から追加、訂正の仕様変更が多すぎた」、「仕様書の出来が悪い」と言い訳をなさる。実はユーザも、要求仕様書の出来が悪かったと反省はしているのだが、要求仕様書の品質向上、受注金額の増加の行為には結びついてゆきにくい。お互いの所為（せい）にし合ってみても良い成果は得られない。

JUAS としてもこの問題は何とかしなくてはいけないと平成 15、16 年度に「ビジネスシステム定義研究プロジェクト」を立ち上げ「エンドユーザによるビジネスシステム定義の進め方」についての報告書を作成し、ユーザ自らがこの問題解決をしようと立ち上がった。その報告書の中には 10 種類のドキュメントを提示し、要求仕様書の書き方の概要を示した。

しかし、その中で詳細な要求仕様はフローで書いてもよいし、HIPO で書いても良いとなっている。「もっと良い方法はないのか」とその後も模索を続けていたところ、偶然にも清水吉男先生の USDM 方式を見つけ出した。ただし、この USDM 方式は組み立てソフトウェアの開発に主として活用されているので、ビジネスシステムの仕様書の作成にこれだけでよいかどうかは、検討する必要がある。

JUAS には 42 種類の研究会･フォーラム・プロジェクトがあり、そこでは、レベルの高い議論がなされている。その中の知恵者を集めたこのプロジェクトを立ち上げ「グチから知恵へ」をトライしたいとこのプロジェクトを始めたわけである。

ユーザとベンダーは協力してこの要求仕様書の明確化に挑戦せねばならないので、この

プロジェクト名を UVC（User Vender Collaboration Project）と名づけた。RFP プロジェクトと普通の名前にしなかったのは、今までの殻を破った発想をして欲しいとする願望の表れである。

結論的にはこのプロジェクトにより以下の知見が得られたと思っている。

- ①機能、②非機能、③データ属性の記述、④ユーザビリティの 4 種類の情報を、要求仕様書に盛り込むこと。
- 機能は機能仕様と理由を分けて書くこと、機能仕様には追い番号をつけ機能階層図を作成する。理由を分けて書くことにより「何故このような仕様になったのか」が明確になるとともに、仕様作成者の意思が正しく伝達される。
- 仕様は仕様番号単位にカウントできるので、仕様変更率＝仕様確定後に仕様変更した仕様数÷総仕様数が計算できる。これを元にユーザ側とベンダーとの間で仕様変更マネジメントを実施する制度を普及したい。
- 機能仕様は追番が付加されてあること、すべての仕様情報、プログラム情報はシステム内に保存されてあること、などの条件が整ったので、仕様からプログラムまでの関連情報の追跡管理が可能となる。
- 機能仕様からデータ構造が組み立てられる。概念データベース、あるいは論理データベースの作成が可能になる。仕様を図で表し、データ要素間の関係を追究・分析することにより仕様の曖昧さ、不十分さの解消が可能になる。
- 非機能仕様についても経験者が議論し、要求仕様へのまとめ方が明確になった。
- ユーザビリティへの要求記述は重要であるが、ユーザビリティを正しく要求仕様書に記述しなさいと指導しているガイドも今まではない。新しく発案したユーザビリティを確認できる 3 手法の活用は比較的簡単である。ユーザビリティを重視するシステムは開発手法そのものを変えたほうがよいことも今回の研究で判明したので JUAS-UCD モデルを創案した。

おおくの関係者の皆様のご努力により、要求仕様書のベストプラクティスが誕生できたと考えている。「問題があれば解決策あり、解決策を考えるところに創造の喜びあり」であることを座右の銘としているが、このプロジェクトを実施でき課題解消策が見つかり、ノウハウが蓄積できたことの喜びを感じると共に関係者のご努力に感謝している。

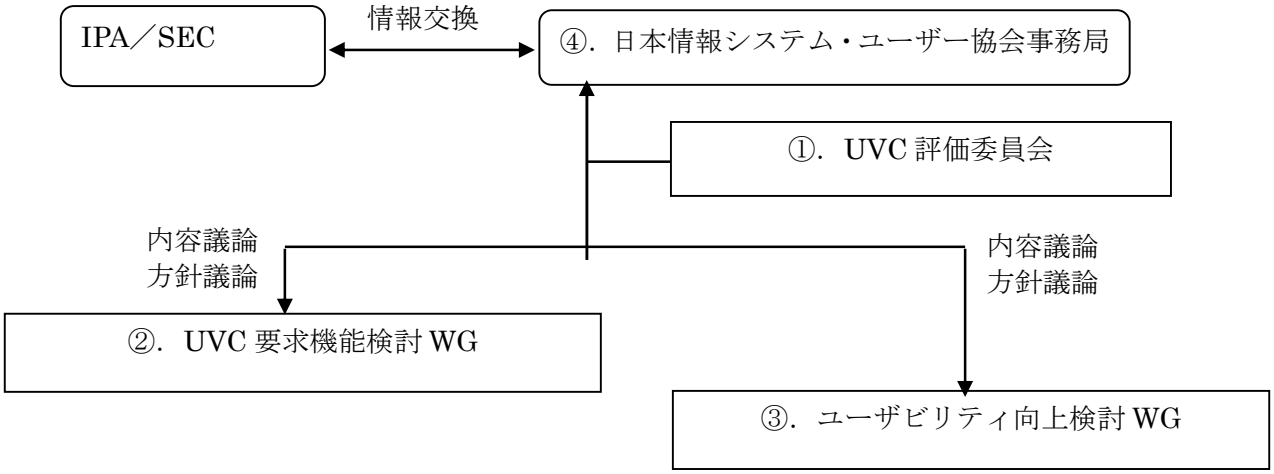
何か動けば何かが得られる。そして世の中が変わる。

専務理事　細川泰秀

2007 年 3 月末日

2. 事業体制

専門家による UVC（User Vender Collaboration）評価委員会と UVC 要求機能検討 WG、ユーザビリティ向上検討 WG を結成した。



①．UVC 評価委員会

目　　的：キーポイントでの意見交換、アドバイスを受ける。②UVC 要求機能検討 WG と③ユーザビリティ向上検討 WG の成果を客観的な視点で組み合わせ、評価する。

開催回数：２回／年

委　　員：全９名（五十音順、敬称略）

玉置　彰宏	東京情報大学（座長）
飯島　淳一	東京工業大学、経営情報学会
大森　正明	東レ株式会社
後藤　将夫	日鉄日立システムエンジニアリング株式会社
清水　吉男	システムクリエイツ株式会社
宗　雅彦	株式会社サイクス
浜田　達夫	株式会社ＪＡＬインフォテック
松本　健一	奈良先端科学技術大学院大学
横山　真一郎	武蔵工業大学

②．UVC 要求機能検討 WG

目　　的：要求機能の検討および企画を行う。

開催回数：９回／年

委　　員：全 13 名（五十音順、敬称略）

玉置　彰宏　東京情報大学（座長）

石井　信也　株式会社テプコシステムズ　電力システム第１本部本部長付
岩佐　洋司　住友電気工業株式会社　情報システム部主幹
太田　忠雄　株式会社ジャステック　常務取締役　営業本部本部長
菊島　靖弘　株式会社アイネス　金融システム本部副本部長
島谷　二郎　東京海上日動システムズ株式会社　代表取締役専務
須斉　智孝　KDDI 株式会社　情報システム本部　システム企画部
　　標準化推進グループ課長補佐
中村　友昭　新日鉄ソリューションズ株式会社　技術本部
中村　伸裕　住友電気工業株式会社　情報システム部　システム技術グループ
　　グループ長
南　悦郎　　新日鉄ソリューションズ株式会社　技術本部
森下　哲成　株式会社リクルート　FIT　システム基盤推進室
　　AP ソリューショングループ
湯田　聖　　丸文情報通信株式会社　管理統括室　室長
横山　透　　東京ガス株式会社　IT 活用推進部 IT 活用企画Ｇマネージャ
事務局　：細川　泰秀　（社）日本情報システム・ユーザー協会　専務理事
　　角田　千晴　（社）日本情報システム・ユーザー協会　事業部長
　　近田　敦子　（社）日本情報システム・ユーザー協会　教育事業部マネージャ

③ユーザビリティ向上検討 WG

目　　的：企業情報システムでのユーザビリティの向上対策を明確化する。
開催回数：９回／年
委　　員：全 17 名（五十音順、敬称略）

黒須　正明　　独立行政法人　メディア教育開発センター教授（座長）
長尾　和洋　　ソニー生命保険株式会社　広報部広告宣伝課主事
内野　栄策　　東京海上日動システムズ株式会社
　　超保険ソリューションサービス部ソリューションデザイナー
遠藤　憲治　　株式会社損保ジャパン・システムソリューション
　　保険システム第一部保険設計グループ
岡野　　猛　　三井不動産株式会社　情報システム部システム開発グループ長
長田　尚丈　　日本通運株式会社　IT 推進部 IT 業務システム専任
奥田　泰彦　　東京電力株式会社　システム企画部流通システム企画グループ
葛城　　周　　東レ株式会社　システム企画開発部
唐橋　哲也　　株式会社損保ジャパン・システムソリューション
　　システム基盤第一部第一グループ

高橋　　明　　株式会社ビーブレイクシステムズ　取締役
永田　孝哉　　伊藤忠テクノソリューションズ株式会社　情報システム部
根岸　吉司　　株式会社 DNP 情報システム
　　システム技術本部技術サポート部シニアエキスパート
福井　恵理　　株式会社アイネス　技術開発本部ソリューションビジネス開発部
福島　　陽　　新日本製鐵株式會社
　　業務プロセス改革推進部システム企画第二グループマネジャー
松本美千代　　株式会社ジャステック　製造本部
山下　貴男　　株式会社デジタルリンク　システム・ソリューション第１部　主任
横井　　朗　　株式会社ビーブレイクシステムズ　プロジェクトマネジャー
事務局　：細川　泰秀　（社）日本情報システム・ユーザー協会　専務理事
　　各務　京子　（社）日本情報システム・ユーザー協会　会員事業部

④日本情報システム・ユーザー協会事務局
担当者　：細川　泰秀　（社）日本情報システム・ユーザー協会　専務理事（責任者）
　　角田　千晴　（社）日本情報システム・ユーザー協会　事業部長
　　近田　敦子　（社）日本情報システム・ユーザー協会　教育事業部
　　マネージャ
　　各務　京子　（社）日本情報システム・ユーザー協会　会員事業部

3. 活動内容

図表 0-1 で、各回の議論テーマの概要を示す。

図表 0-1　活動内容

回	日付	人数	議論のテーマ
第１回	6 月 26 日(月) 16:00-19:00	22 名	・ 当プロジェクト趣旨説明自己紹介 ・ 当プロジェクトの方針について議論 ・ 「情報システムの信頼性向上のための取引慣行・契約に関する研究会」のご報告　経済産業省　福田哲平様
第２回	7 月 18 日(火) 16:00-19:00	21 名	・ （テーマ２　利用部門、ＩＴ部門の責任者によるシステムコンセプトの確認内容の明確化。）「リクルートにおけるプロジェクト編成についての考え方と必要資料」 リクルート森下様 ・ （テーマ１　暗黙知の整理）QFD についての説明＆議論 玉置先生 ・ 今後のスケジュールについての確認
第３回	8 月 28 日(月) 16:00-19:00	39 名	・ 「要求を仕様化する技術」について　システムクリエイツ　清水様よりご講演 ・ 質疑応答 ・ テーマ１「暗黙知の整理」について　問題提起＆議論、次回までの宿題説明

第 4 回	9 月 20 日(水) 16:00-19:00	37 名	・ 「暗黙知の整理」新日鉄ソリューションズ様 ・ 「暗黙知の整理について全員にて議論」 ・ ①テーマ２：利用部門、IT 部門の責任者によるシステムコンセプトの確認内容の明確化。 ・ ②テーマ３：要求仕様の定義方法 ・ ③テーマ４：要求仕様→設計→プログラムの仕様をトレースできる方法についての問題提起
第 5 回	10 月 24 日(火) 16:00-19:00	26 名	・ ①利用部門、IT 部門の責任者によるシステムコンセプトの確認内容の明確化。テプコシステムズ　石井様、東京ガス　横山様 ・ ②要求仕様の定義方法　新日鉄ソリューションズ南様、中村様 ・ ③要求仕様→設計→プログラムの仕様をトレースできる方法　住友電工　岩佐様、中村様 ・ 次回「DB の定義方法」テーマについての課題提案
第 6 回	11 月 28 日(火) 16:00-19:00	37 名	・ 「DB の定義方法についての取り組み」KDDI　様　住友電工　様 ・ 「画面、帳票のデザインとユーザビリティ」※ユーザビリティ部会リーダー長尾様よりご紹介 ・ 次回のテーマについて
第 7 回	12 月 19 日(火) 16:00-19:00	39 名	・ 「プロジェクトを始めるときに要件定義の対象としなければいけない事項」IPA　菊島様 ・ 「ユーザーが理解しやすい業務設計モデル」 丸文情報通信　湯田様 ・ 次回のテーマについて・UVC 評価委員会について
第 8 回	1 月 23 日(火) 16:00-19:00	36 名	・ 「品質要求の定義方法（非機能要件)。品質を確保するために、ユーザーとベンダー両者の確認項目を整理する。」東京海上日動システムズ　島谷様 ・ 「非機能要求と基盤設計のプロセスについて」新日鉄ソリューションズ　システム研究開発センターシステム基盤研究部　中村 賢亮（まさあき）様 ・ 「ジャステック流妥当性確認の一つのやり方として、品質特性(例：JIS-X0129)の適用した例」ジャステック　太田様
第 9 回	2 月 23 日(金)、24 日（土）合宿	32 名	・ 「品質要求の定義方法。品質を確保するために、ユーザーとベンダー両者の確認項目を整理する。」丸文情報通信　湯田様 ・ UVC 研究プロジェクト報告書原稿を元に追加議論 ・ 図書館管理システム　清水方式での要求仕様書検証（新日鉄ソリューションズ中村様より）
第 10 回	3 月 26 日(月) 16:00-19:00	34 名	・ 報告書内容の最終確認と議論

１．このプロジェクトの目的と背景

このプロジェクトの目的

このプロジェクトの目的は、「情報システムを開発するに当たって、要求仕様書をいかに書くか」を明らかにすること、つまり要求仕様書についての「How」を明確にすることである。

なお我々が所属する組織は日本情報システム・ユーザー協会（JUAS）の研究会であるという位置づけから、このプロジェクトで明確にしようとする要求仕様書はビジネス・アプリケーションに関わるものに限定する。つまり組み込み型のソフトウェアなどは対象としていないことを、最初に明確にしておきたい。

上記目的設定の背景

このプロジェクトの今年度の目的をこのように定めた背景として、以下のものを挙げることができる。

- JUAS はこれまで、要求仕様書に「何を記述すればよいか」について研究をした実績がある。しかし「要求仕様書をいかに書くか」の研究は、まだ未着手であった。
- さらに独立行政法人情報処理推進機構（IPA）のソフトウェア・エンジニアリング・センター（SEC）の活動の中に、要求仕様書として何を記述するのかについての研究が含まれている。
- JUAS の調査によると、情報システムの開発で納期の遅延、予算の超過、品質に対する不満足が発生する割合が高い。これらの不満足発生の原因に、要求仕様書作成に関わる問題が指摘されている。
- やはり JUAS の調査によると、情報システムの開発を発注したとき、仕様の定義不充分のまま発注したこと、要求仕様条件を明確に提示しなかったことの二点が反省点の上位に挙げられている。
- 逆に要求仕様書を適切に作成することで、納期遅れのリスクの減少、プロダクトの品質の向上が期待できる。

以下で、上記の内容を少し詳細に見ておきたい。

背景－1

これまで JUAS は、要求仕様書をはじめとする情報システムの開発に関わるいろんな問題について研究を行ってきた。その成果は、それぞれの報告書に記載されている。その中に、要求仕様書として「何を記述すればよいか」についてまとめたものがある[JUA04]。

ただこの時点では、要求仕様書の書き方について深い分析を行っていない。したがって JUAS としてはこの研究の後を受けて、要求仕様書をいかに書くのかについての回答を提示

する必要性を認識していた。これが JUAS としての、この研究会を実施する 1 つの背景になっている。

以前の研究とこの研究との関連については、第 13 章で再度述べる。

背景－２

IPA／SEC では、2005 年度より継続して「開発プロセス共有化部会」を活動させている。その活動の一部としてユーザとベンダーが合同で、情報システム開発の「超上流工程」のあり方の検討を行っている。

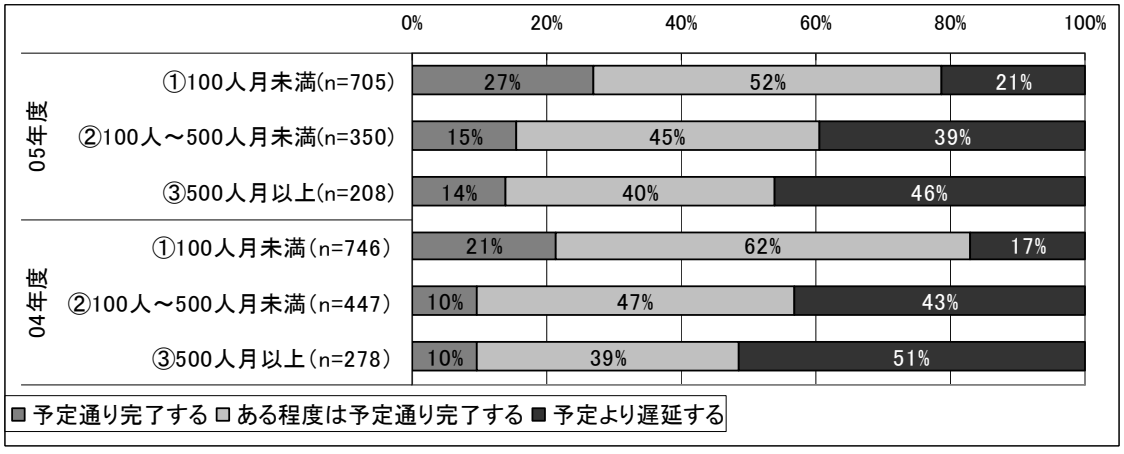
ここでの研究内容は単に要求仕様書の作成だけに関わるものではなく、もっと幅が広く、奥行きも深い。しかしその活動の一部に「要求仕様書とは何か」、「要求仕様書には何を書くべきか」といった、要求仕様書作成に関わる「What」の部分が含まれている。

したがって我々のプロジェクトはその成果としての「What」の部分を受けて、それに続く「How」の部分に特化する形としたい。これが、ここで「要求仕様書をいかに書くか」について考えようとする理由の 2 つ目である。

背景－３

次に、要求仕様書そのものに関わる問題からこの背景を探ってみたい。

JUAS では、1994 年より毎年ユーザ企業 IT 動向調査を実施している。この調査はアンケート調査とインタビュー調査から構成されている。その 2006 年版のアンケート調査（調査は 2005 年秋に実施）で、システム開発における工期、予算、及び品質についての満足度を聞いている。その結果を図表 1-1 から 1-3 に示す。



図表 1-1　システム開発における工期についての状況（[JUA06b]より）

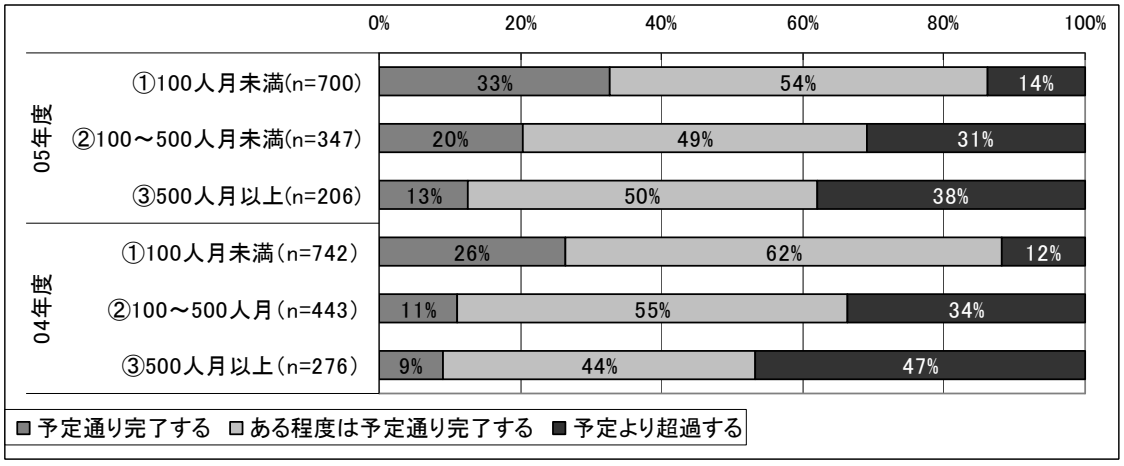
開発する情報システムの規模によって「予定通り完了する」、および「満足」の割合、あるいは逆に「予定より遅延する」、「予定より超過する」、および「不満足」の割合は異なっ

ている。しかしいずれの場合も、うまくいっている割合は少ない。

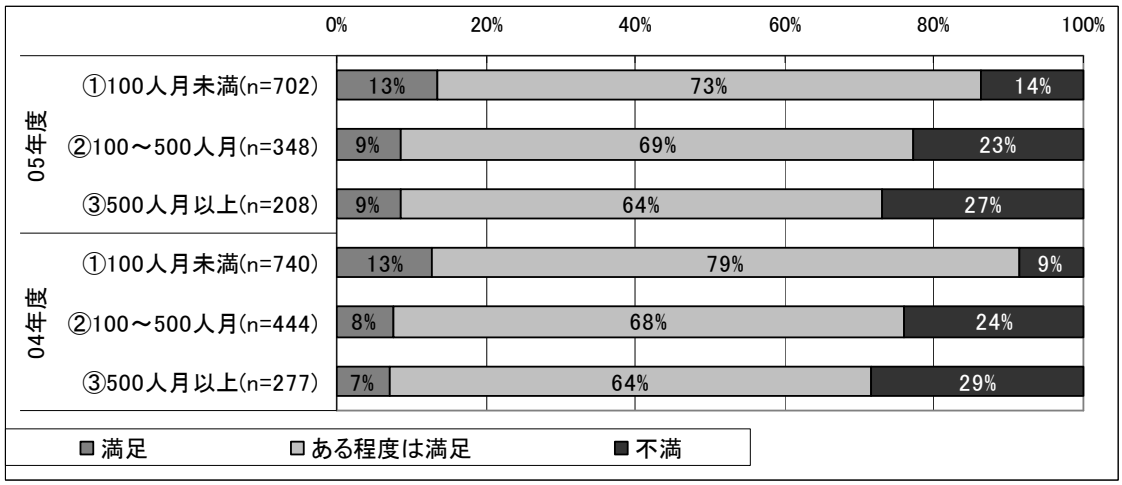
現代のソフトウェア危機の症状は

- スケジュール遅れ
- 開発予算の超過
- 品質面での問題

の 3 つの中の 1 つ以上が顕在化し、場合によるとそれらが相まって、特に米国あたりではソフトウェア開発プロジェクトが当初予定したプロダクトを完成できないまま解散させられるという形で表れる[TAM03]。JUAS の調査で、プロジェクトの解散にまでは至らなくても、日本国内でもソフトウェア危機の症状の一部が顕在化していることが明確になっている。



図表 1-2　システム開発における予算の状況（[JUA06b]より）



図表 1-3　システム開発における品質の状況（[JUA06b]より）

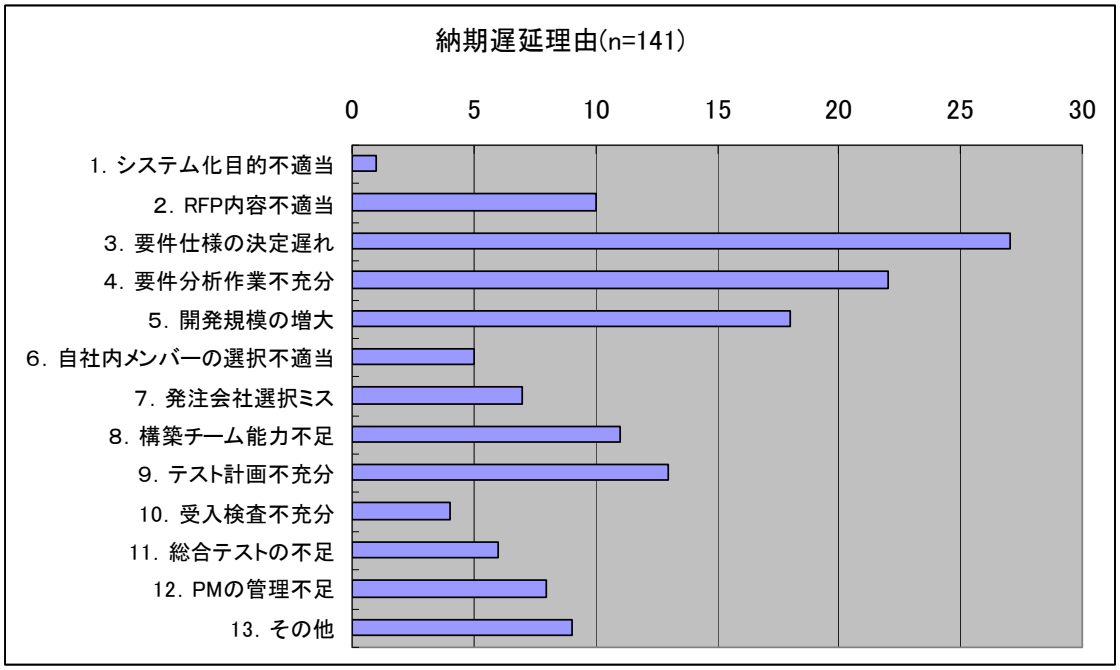
それでは、このソフトウェア危機の現象が起きる理由は何か。これについても JUAS は調査を行っている。「ソフトウェアメトリクス調査」である。

この調査は 2005 年から開始されたが、その 2006 年の調査報告（調査は 2005 年秋に実施）に納期、つまりスケジュール遅れについてその理由を聞いた結果が掲載されている[JUA06a]。その結果を図表 1-4 に示す。

この図の中、「２．RFP 内容不適当」と「３．要件仕様の決定遅れ」、「４．要件分析作業不充分」の３つがが何らかの形で要求仕様書の作成に関わる項目である。グラフから明確なように、この中の「3」と「4」は件数が多く、これに「2」を加えた 3 つの要件で 59 件、全体に占める割合では 42%にもなっている。

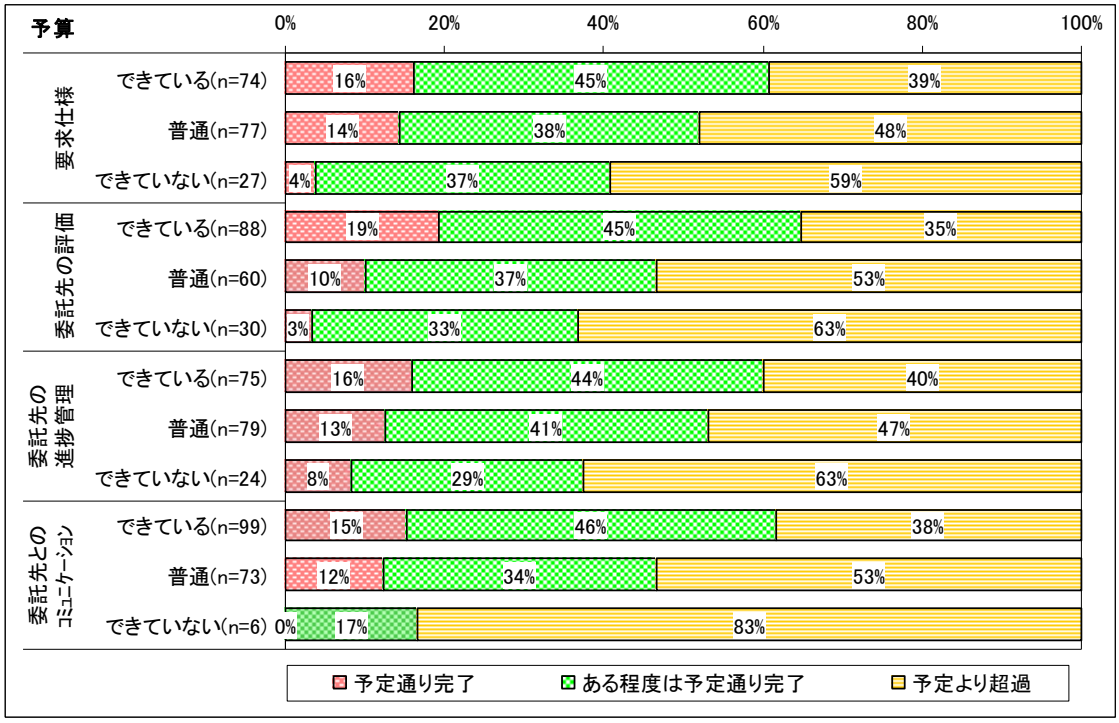
つまり、スケジュール遅れの原因の少なくない部分が要求仕様書の作成に関わりを持っているということを指摘しておきたい。言い換えると、要求仕様書を的確に作成することができると、スケジュール遅れのある部分は解消することができる。

これが、ここで「要求仕様書をいかに書くか」について考えようとする理由の 3 つ目である。

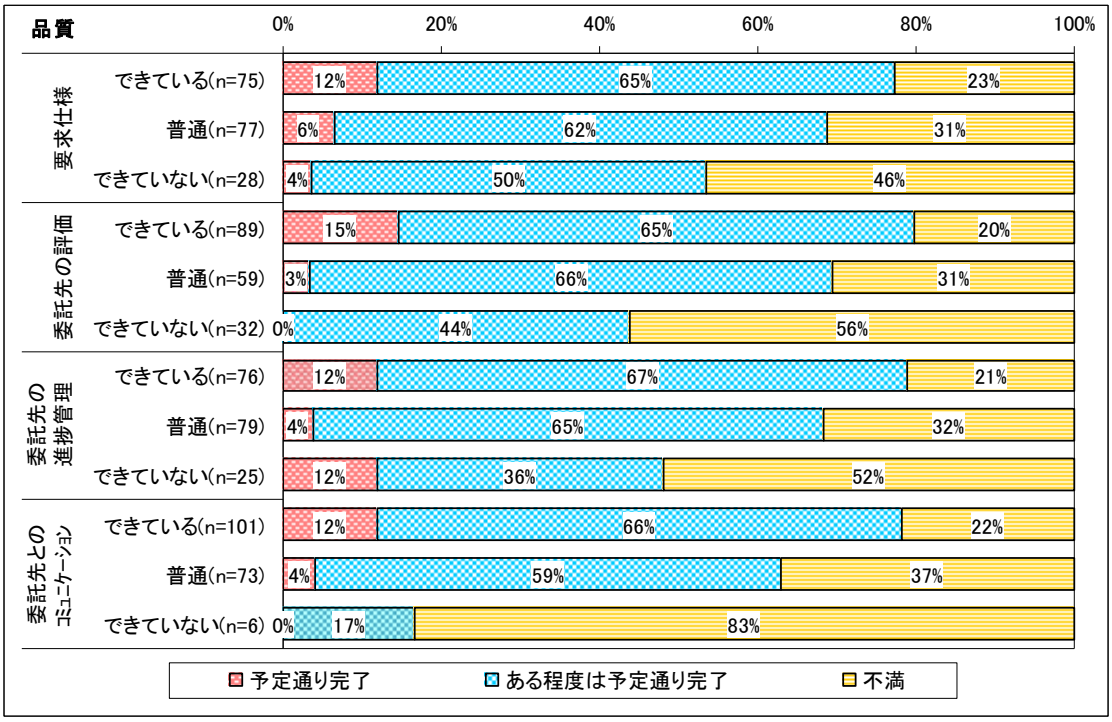


図表 1-4　納期遅延の理由（[JUA06a]より）

さらに要求仕様の提示を適切に行うことで、予算を超過させないことにも品質の確保にも効果があるとの調査結果がある[JUA07c]。その調査結果を、図表 1-5 と 1-6 に示す。



図表 1-5　予算と要求仕様の提示を含むプロジェクト要件との関係[JUA07c]



図表 1-6　品質と要求仕様の提示を含むプロジェクト要件との関係[JUA07c]

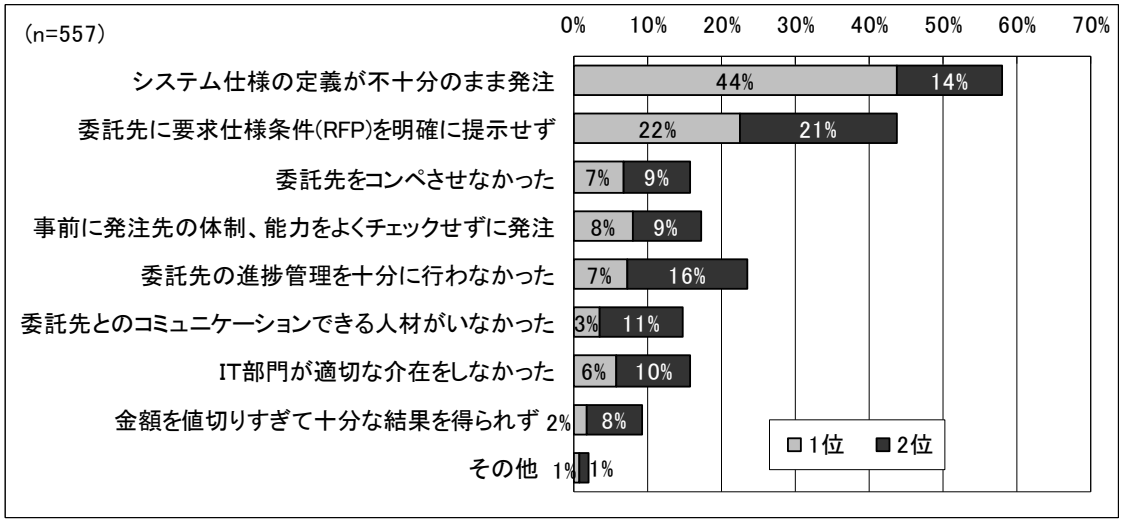
背景－４

上記と同じアプローチになるが、やはり 2006 年版の企業 IT 動向調査のアンケート調査[JUA06 b]に、ソフトウェア開発を発注した後の「発注者としての反省点」を聞いた結果がある。それを図表 1-7 に示す。

この設問は、あらかじめ用意した 9 つ（「その他」を含む）の項目から第一位と第二位の 2 つを選んで回答して貰う形になっている。

ここでも要求仕様書に関わる 2 つの要件（「システム仕様の定義が不充分なまま発注した」と「要求仕様条件（RFP）を明確に提示しなかった」）が、他を圧した高い数値を獲得していることが分かる。ソフトウェア開発に関わる反省点の上位に要求仕様書作成に関わる項目があるということは、要求定義の不完全さ、不充分さがシステム開発の失敗に結びつくという事態を今後ユーザは避けたいと考えていることを示している。

これが、「要求仕様書をいかに書くか」について考えようとする理由の４つ目である。



図表 1-7　発注者としての反省点（[JUA06b]より）

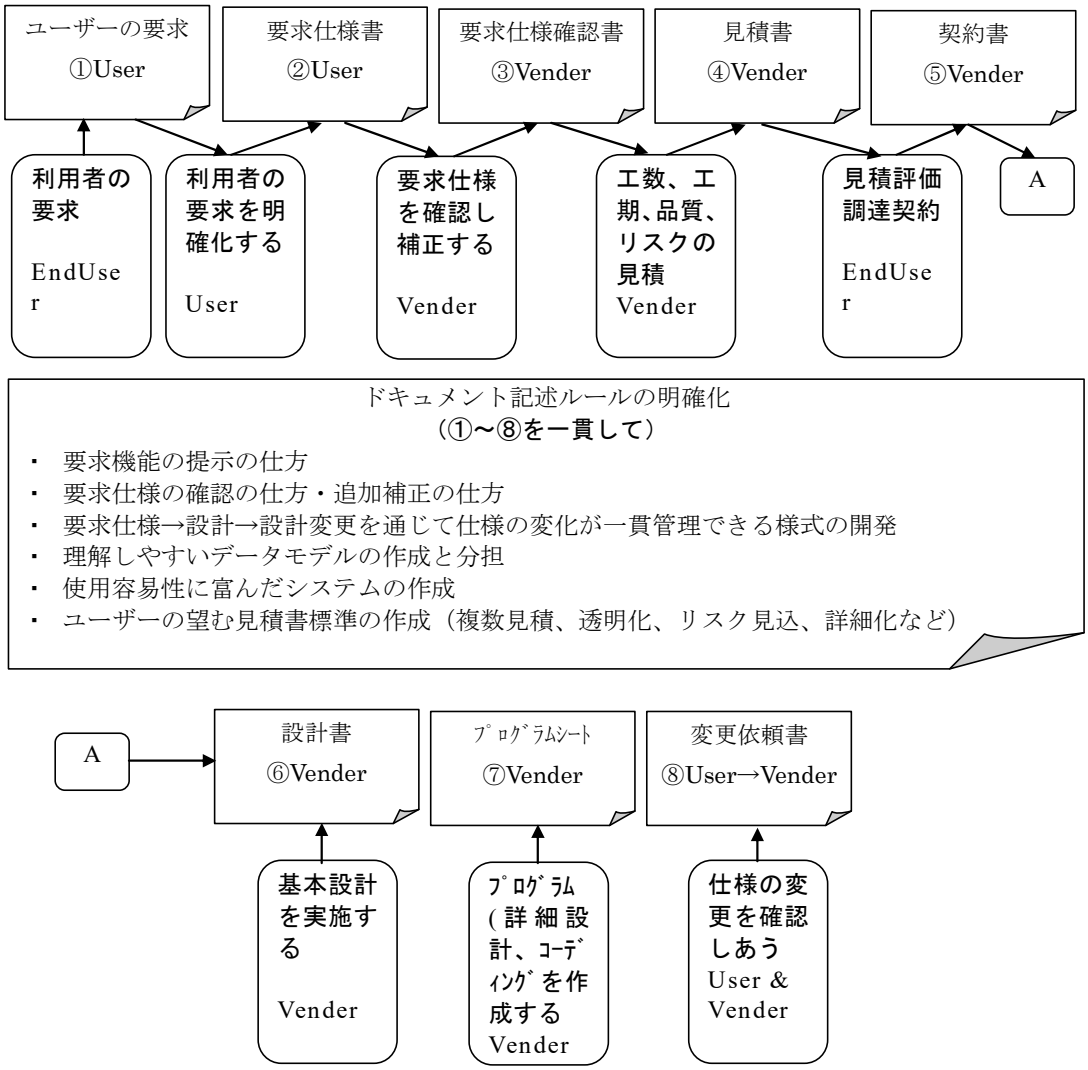
背景－５

次に立場を変えて、的確な要求仕様書を作成することで、ユーザにとってどんな利点があるのかを考えたい。

図表 1-8 は、情報システム開発の作業の流れを示したものである。この中で要求仕様書は「利用者の要求を明確にした」2 つ目の成果物として位置づけされている。さらに言うまでもないことであるがこの図から、構築される情報システムはこの要求仕様書に記述されたものを基に構築されるものであることが分かる。

このことからだけでも適切な要求仕様書の重要性が分かるが、さらにユーザが適切に要求仕様書を作成することで、次のような利点を受けることができる。

- 要求仕様書が適切に作成されていると、仕様の漏れが少ない。そのためベンダー側の作業である設計以降の工程で仕様確認のための余分な作業や作業の手戻りが減少し、開発の生産性向上が期待できる。
- 要求仕様書が適切に作成されていると、仕様から設計、プログラミングのトレースがより的確にできるようになる。そのため、仮に仕様変更が入っても、それに対応するための工数が少なくなる。
- 要求仕様書が適切に作成されていると、テストケースの洗い出しがより適切にできるようになり、テスト実施の効率が向上する。
- 設計作業とテストケース設定の作業を、並行して進めることができる。



図表 1-8　情報システム開発作業の流れ

- ベンダー側のリスクが減少する。それにともないベンダーの開発費用の見積もりがより適切なものとなる。
- 納期遅れのリスクが減少する。
- プロダクトの品質の向上が期待できる。

これが、ここで「要求仕様書をいかに書くか」について考えようとする理由の 5 つ目である。

しかし「How」を考える上で「What」は不可欠な要件であり、必要であれば我々なりの「What」を明確にする作業の実施を阻むものではないことを、ここで明確にしておきたい。

また我々は、「要求仕様書を作成するのはユーザ企業である」との立場で、要求仕様書の書き方についての研究を行う。これと異なる対応方法については、第 13 章で議論する。

この報告書の構成

この報告書は、以下のような構成になっている。

この第 1 章では、この研究の目的と背景について述べた。

次の第 2 章では、要求仕様書に関わる問題として今どのようなものがあるのかを洗い出し、それらをどのように我々が解決しようとしているのかをまとめる。

第 3 章では、我々が要求仕様書作成のひな形として採用しようとしている「USDM（Universal Specification Description Manner）表記法」の概要を述べる。

第 4 章では、情報システムを開発する作業の中、要求仕様書作成に関わるいくつかの作業と要求仕様書作成の作業そのものとの関係、他の作業での留意事項などについて述べる。

第 5 章では、要求仕様書を作成するユーザ部門と情報システムを開発する SE の間で、いかにして知識を共有するのかという問題を議論する。

第 6 章では、我々の要求仕様書として第 3 章で述べた USDM 表記法に何を付け加え、機能要求と呼ばれるものをどのように記述するのかという問題を議論する。

第 7 章では要求仕様書作成についてのユーザ部門と IT 部門の関係を、第 8 章は仕様を要求仕様書から設計書、プログラムまでトレースする方法を、さらに第 9 章ではデータベースの定義方法を、それぞれ述べる。

第 10 章では我々の研究会と並行して行われてきたユーザビリティの研究会の成果を要約し、第 11 章では非機能要求と呼ばれるものをどう定義するのかについて述べる。

第 12 章で、この報告書で述べてきた要求仕様書を 1 つのサンプル（図書管理システム）で例示し、評価を行い、併せて得られた知見について述べる。

第 13 章では、この研究の成果を実際の開発に適用する方法について議論する。

そして最後の第 14 章で、残された課題をまとめる。

参考文献

[JUA04] 「エンドユーザによるビジネスシステム定義の進め方 平成 15 年度ビジネスシス

テム定義研究プロジェクト報告書」、（社）日本情報システム・ユーザー協会、平成 16 年 4 月.

[JUA06a] 「ユーザー企業ソフトウェアメトリクス調査 2006」、（社）日本情報システム・ユーザー協会、2006 年.

[JUA06b] 「企業 IT 動向調査 報告書 2006 年版」、(社) 日本情報システム・ユーザー協会、2006 年.

[JUA07c] 「企業 IT 動向調査 報告書 2007 年版」、(社) 日本情報システム・ユーザー協会、2007 年.（未刊行）

[TAM03] 玉置彰宏著、「ソフトウェア工学の勧め 第 1 章 ソフトウェア工学とは」. この原稿は以下の URL からダウンロードできる。

http://www.rsch.tuis.ac.jp/~tamaki/software_engineering/Chap_01.pdf

２．要求仕様書に関わる問題とその解決方法

要求仕様書に関わる問題

要求仕様書が適切に作成されないことで、情報システムの開発スケジュールの遅延が起きる可能性があることを第 1 章で指摘した。またに要求仕様書を適切に作成することで、開発予算の超過や品質の低下を防止することに効果があることも、第 1 章で示した。つまり今のソフトウェア危機の原因のいくつかは、要求仕様書の問題に行き着くと考えて良い。

それでは、要求仕様書に関わる問題にどのようなものがあるのか。まずそれを、いくつかの調査報告書や文献などから集めてみたい。その上で我々はそれらにどう対応しようとしているのかの全体像を、ここで示してみたい。

個別の問題点

要求仕様書作成に関わる個別の問題は、各種の調査報告書や文献などに多く報告されている。それらを、重複を恐れずに、目につくものを挙げると、以下に示すようなものになる。

1. 開発するべき情報システムの範囲やスコープの問題
   - ① スコープが不明確（[RYU06]）
   - ② 開発の進展と共に仕様が膨張する（[IPA05]）
2. 仕様の記述に関わる問題
   - ① 仕様の記述に漏れがある（[SIM05]）
   - ② 仕様の記述に抜けがある（[JUA05]）
   - ③ 要求の背景が不明確（[RYU06]）
   - ④ システム仕様の定義が不充分なまま発注した（[JUA06b]）
   - ⑤ 要求仕様条件を明確に提示しなかった（[JUA06b]）
   - ⑥ 顧客の当たり前要求が記載されていない（[RYU06]）
   - ⑦ 具体的な要求が欠如している（[RYU06]）
   - ⑧ 記述の方法が不明確（[RYU06]）
3. 仕様の解釈に関する問題
   - ① 読み手側の仕様の誤解釈[SIM05]
4. 仕様の衝突・矛盾の問題
   - ① 要求項目同士に矛盾がある（[RYU06]）
   - ② 仕様間の衝突の発見が遅れる[SIM05]
5. その他の問題
   - ① 要件仕様の決定が遅れる（[JUA06a]）
   - ② 担当者が変わると仕様も変わる（[JUA05]）

仕様の記述に関わる問題の解決方法

要求仕様書についての問題の中で、仕様の記述に関わる問題が種類としてやはり一番多い。

これらは、要求仕様書の書き方そのものの中で解決したい。具体的には「３．要求仕様書とは」の中で述べるが、我々の要求仕様書の書き方として「USDM 表記法」を採用することにした。この方法を採用することで、仕様の漏れ、抜けも含めて、ここに書かれた問題は解決可能と我々は考える。

さらに「USDM 表記法」によって、仕様間の衝突や矛盾も解決についての問題も併せて解決可能と我々は考えている。それだけでは対処できそうにない場合の対応方法は、「６．要求仕様の定義方法」で述べる。

情報システムの範囲やスコープの問題への対応

スコープの問題への対応については、次の方法で対応する。

要求仕様書作成のフェーズの前に「ビジネス検討」と呼ぶフェーズを置いて、そこでこれから開発しようとする情報システムについて、情報戦略上の位置づけ、この情報システムの範囲・領域などを確認し、それを受けてこの要求仕様書の作成を始める。

また要求分析移行の作業で仕様の膨張が生じたときにも、この「ビジネス検討」での結論に戻って、その膨張を受け入れることが妥当かどうかを判断する方式を採用する。

その「ビジネス検討」のフェーズについては、「４．要求仕様書作成に関わる作業の流れ」で議論する。

仕様の解釈に関する問題への対応

要求仕様書を書いた人たちと後でそれを読む人たちの間で、そこに書かれていることについての解釈の不一致が起きることがある。これは暗黙のルールの存在と併せて、それぞれの業界に固有の用語が存在することなどがその原因となることが多い。

この問題の解決については、「５．知識共有の方法」で議論する。さらにその結論を先取りすれば、「３．要求仕様書とは」で結論を出した「USDM 表記法」による要求仕様書の作成に加えて、要求仕様の一部として概念データモデルを作成する。さらに専門用語などについて、要求仕様書の一部にそれを記述する方法を提示する。

概念データモデルの作成については、「９．データベースの定義方法」で議論する。

その他の問題への対応

「担当者が変わると仕様が変わる」とか、「要求仕様の提示が遅れる」といった問題は、ユーザ側の組織の問題と捉えてみたい。その上で、「７．ユーザ部門と IT 部門の役割」の中で解決方法の議論を行う。

上記問題提起の中には含まれていなかったが、仕様のトレースの問題への対応は、「８．要求仕様を設計からプログラムまでトレースする方法」の中で、データベースの定義方法

は前述の通り「９、データベースの定義方法」で、また情報システムからの出力のデザインや使いやすさを実現するための問題については、「10. 画面／帳表のデザインとユーザビリティ」の中で、それぞれ議論する。

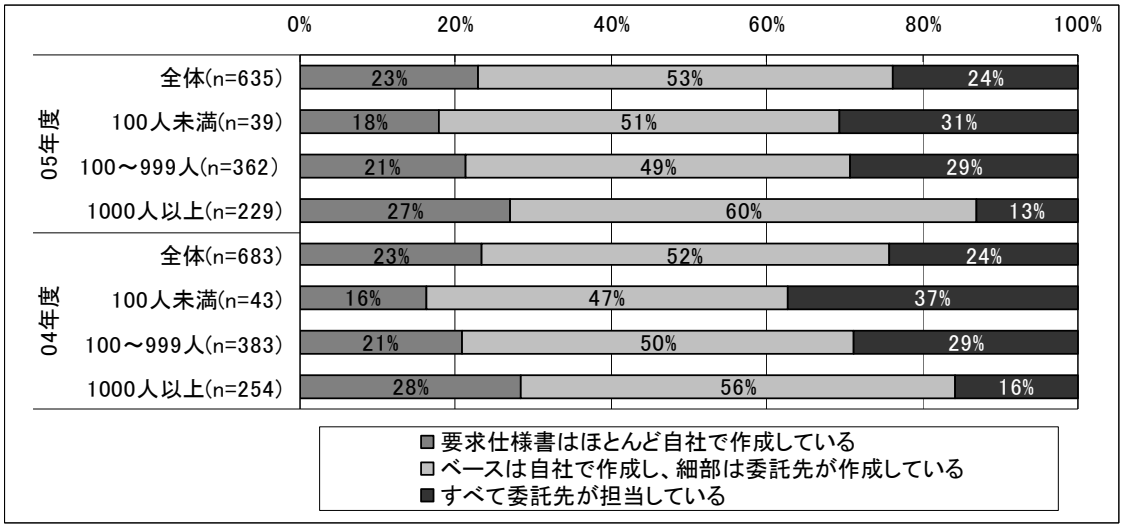
さらに要求の中の「非機能要求」と呼ばれるものの記述方法は、「11. 非機能要求の定義方法」で議論する。

最後に残された問題、つまり「要求仕様書の表現方法」については、SE にとっての文章作成術の書籍[HUK05]を使用することで、その解決としたい。

要求仕様書は誰が作成するのか

最後に、要求仕様書は誰が作成するべきなのかという問題について、問題提起をしておきたい。

JUAS の企業 IT 動向調査の 2006 年版に、図表 2-1 に示す報告が記載されている。



図表 2-1　要求仕様書作成における役割分担（[JUA06b]より）

この調査結果によると 2005 年度で、全体の約 4 分の 1（24%）が、「全て委託先が担当している」と答えている。従業員の数が 1000 人以上の大企業でも 8 分の 1 以上（13%）が、従業員数 100 人未満の小企業では 3 分の 1 近く（31%）が、「全て委託先が担当している」と答えているという現実を、指摘しておきたい。

委託先にどんな会社があるのか、ユーザ企業と委託先の関係がどうかといった個々の事情は、ここでは分からない。しかし一般的にいえることは、要求仕様書は基本的にユーザ企業の責任で書くべきであるということである。

後述するが我々は、「要求仕様書はユーザと SE が共同で書くべき」とする立場を取る。図表 2-1 で言えば、「ベースは自社で作成し、細部は委託先が作成している」の回答になる

だろう。

我々はそのような立場で、いかに要求仕様書を書くのかという議論を展開してゆきたい。

これらの議論を通して、「全てを委託先に投げて、それをもって要求仕様書の作成は完了」という形は、避けていただきたいと考えている。

参考文献

[HUK05] 日本情報システム・ユーザー協会編、福田修著、「SE を極める 仕事に役立つ文章作成術」、日経 BP 社、2005 年.

[IPA05] 「プロジェクト編成会資料 XX 情報誌リニューアルプロジェクト」、IPA、2005 年.

この資料は独立行政法人情報処理推進機構（IPA）のソフトウェア・エンジニアリング・センター（SEC）にある「事例検索システム」のページ（URL は https://sec.ipa.go.jp/enterprise/index.php?type=s&ccd=n）に、提供社名は「B 社」、資料名は「プロジェクト編成会資料」として登録されており、そこからダウンロードすることができる。(実際にダウンロードするためには、IPA にユーザ登録する必要がある。)

[JUA05] 「システム・リファレンス・マニュアル（SRM)」、(社) 日本情報システム・ユーザー協会、2005 年 9 月.

[JUA06a] 「ユーザー企業ソフトウェアメトリクス調査 2006」、(社) 日本情報システム・ユーザー協会、2006 年.

[JUA06b] 「企業 IT 動向調査 報告書 2006 年版」、(社) 日本情報システム・ユーザー協会、2006 年.

[RYU06] 劉功義、「プロジェクト計画における要求整理方法の提案 プロジェクトマネジメント学会 2005 年度第 2 回研究員会フォーラムでの講演資料より」、PM 学会研究委員会、平成 18 年 3 月 1 日.

[SIM05] 清水吉男著、「[入門+実践]要求を仕様化する技術表現する技術～仕様が書けていますか」、(株) 技術評論社、平成 17 年.

３．要求仕様書とは

我々の立場

既に記したように、我々のこのプロジェクトの目的は「要求仕様書とは何か」、「要求仕様書に何を書けばよいか」を決めることではない。我々の目的は「どうすれば良い要求仕様を作ることができるか」を明らかにすることである。別の表現を使えば、我々の目的は“What”を決めることではなく、“How”を明らかにすることである。

しかしこの章のすぐ後の部分で明らかにするように、要求仕様書に関する“What”は、まだ明確には決まってはいない。あるいはことによると、明確には決められない性格のものかも知れない。しかしこの“What”を抜きにして“How”を論じることはできない。

したがって我々は、我々の本来の目的である“How”を論じる前に、「我々が想定している要求仕様書とはどんなものか」、つまり我々なりの“What”をまず明らかにして、その後で我々の命題である“How”を考えるという段階を踏みたい。

この章は、その「我々が想定している要求仕様書とはどんなものか」という“What”を明らかにする章として位置づける。

要求仕様書とは何か

清水吉男氏によると要求仕様書とは、「これから開発するソフトウェアの『作業のゴールとしての要件』を明らかにするものであり、顧客や開発関係者間でこれから作るものについて合意するための文書である」という[SIM05]。

ソフトウェア開発の過程では、多くの文書類が作成される。要求仕様書はそのごく初期に作成されるものであり、その前は曖昧模糊としていた情報システムへの要求（ニーズ）を初めて文書にしたもの、ということができる。この考え方を図で表現すると、図表 3-1 のようになる。

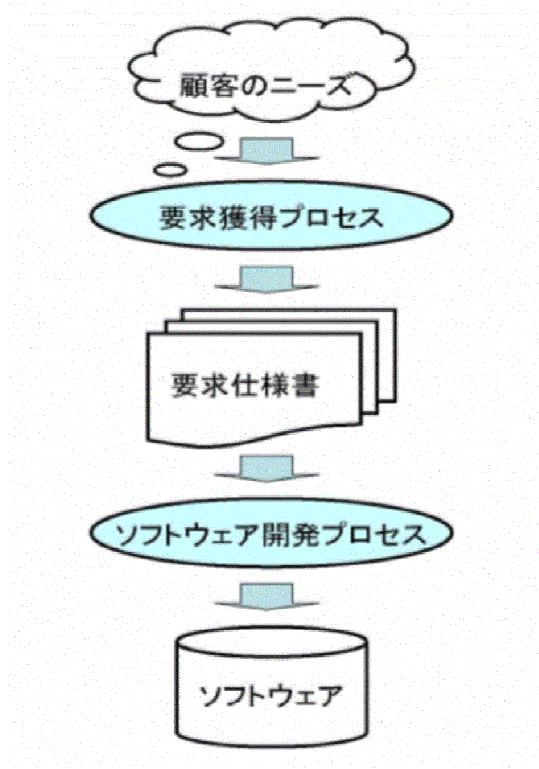
現時点での「品質」についての国際的な公式の定義は、2000 年版の ISO 9000 の規格（ISO 9000：2000）の中にある[ISO00]。それによると品質とは、「本来備わっている特性の集まりが、要求事項を満たす程度」とある。たいへん難しい表現だが、品質に関わるオーソリティの一人であるクロスビーは、これを「ユーザの要求への適合度」と端的に表現している[CRO79]。つまり、「ユーザのニーズに合っている割合が高い製品ほど、品質が高い」というわけである。

ここでクロスビーは「ユーザ」という言葉を使用しているが、最近はこの「ユーザ」という言葉に代えて「利害関係者（または、ステークホールダ）」という表現が使われることが多い。ユーザは利害関係者の一部であるが、全てではない。しかし議論を簡単にするために、以下では「ユーザ」という言葉をそのまま使い続けることにする。

つまりソフトウェアは「ユーザの要求により多く適合しているほど品質が高い」訳であるから、図表 3-1 に基づくと、まず高品質のソフトウェアを作るためには、「要求獲得のプロ

セス」で曖昧模糊としたユーザ（顧客）のニーズを的確に把握し、その結果を明確に要求仕様書に記述することが必要である。その上でその要求仕様書に記述された内容を、「ソフトウェア開発プロセス」を通してソフトウェアに仕上げてゆくことになる。このように見ると、高い品質のソフトウェアを作る上で、要求仕様書はたいへん重要な文書であるということになる。

なお、「ユーザのニーズはビジネス全体から見て常に正しいのか」という、もう一段高い観点からの議論がある。しかし我々は今ここで、このテーマについて議論をすることは避けたい。つまり「ユーザのニーズを我々の情報システム開発の出発点にする」ということを、ここで明確にしておきたい。



図表 3-1　要求仕様書の位置づけ（[SAN94］より）

また清水氏によれば、要求仕様書の内容や書き方は、「ソフトウェア開発プロセス」を担当する組織や SE の適用アプリケーション領域についての知識などによって変わることになるという。つまりソフトウェア開発を担当する組織や SE が対象のアプリケーション領域に詳しい場合、要求仕様書は簡単なもので良い。

このため、要求仕様書に記載される内容の深さは一定していない。その結果、要求仕様書はソフトウェアの開発が完了すると用済みとなるべきである。ソフトウェアの保守などのためにそのソフトウェアの機能を明確に説明するためには、この要求仕様書を基に「機能仕様書」という別の文書を作るべきと清水氏は主張する[SIM05]。

要求仕様書をどう作るのか

要求仕様書の書き方について、IEEE が“IEEE Recommended Practice for Software Requirements Specifications IEEE Std 830-1998”という規格を制定している。

この規格によると、要求仕様書に記載されるべき内容として、次のものをあげている[IEE98]。

1. 機能
2. 外部とのインタフェース（人間、ハードウェア、他のソフトウェアとの）
3. パフォーマンス（スピード、アベイラビリティ、レスポンス・タイム、復旧時間、など）
4. アトリビュート（移植性、正確さ（コレクトネス）、保守の容易性、安全性、など）
5. 設計上の制約

この最初の「機能」に関する要求を「機能要求」、それ以外のものを「非機能要求」と呼ぶ。つまり要求仕様書には「機能」に関する要求に加えて、機能に関わる要求以外のものも「非機能要求」として記載しなければならない。

ここで、「何が機能か」ということについての議論がある。我々は「機能（Function）とは、入力を出力に変換すること」と単純に定義し、この定義に基づいて上記の機能要求と非機能要求の区分けを行っている。

さらにこの規格では、要求仕様書は以下の要件を満たさなければならないとしている[IEE98]。

1. 正確であること（Correct）。
2. 曖昧でないこと（Unambiguous）。
3. 完全であること（Complete）。
4. 首尾一貫していること（Consistent）。
5. 重要さと安定性のためにランク付けされていること（Ranked for importance and/or stability）。
6. 検証可能であること（Verifiable）。
7. 修正可能であること（Modifiable）。
8. 追跡可能であること（Traceable）。

その上でこの規格は、要求仕様書の書き方について 8 種類のひな形を用意している。その詳細の記述はここでは省略するが、あるものは構造化技法時代からのオーソドックスな書き方であり、あるものは新しいオブジェクト指向流のものである。このことから我々は、上記の要件を満たすならば、要求仕様書の具体的な書き方には必ずしもこだわらなくても良いものと理解した。

この IEEE の規格を基に、要求仕様書の具体的な書き方についていくつかの良い本が出版されている。オーソドックスな書き方については秋本芳伸氏らの著書があり[AKI04]、オブ

ジェクト指向流の書き方にはアリスター・コーバーン氏の著書がある[COC01]。いずれも、的確な要求仕様書の記述方法と評価する。1998 年版のもう 1 つ前の 1993 年版の IEEE の規格に基づくものだが、単に要求仕様書の書き方だけでなく、いかにして要求仕様を抽出し、まとめるかということにまで踏み込んだイアン・サマヴィル氏らの著作もある[SOM97]。しかし後述するように、我々は清水氏が提案する方法[SIM05]を我々の要求仕様書の記述方法として取り入れることとした。

要求の種類

秋本氏らは、要求仕様書に書かれる「要求」には 3 つのレベルがあると指摘している。具体的には、以下の通りである[AKI04]。

「業務要求」: システム開発の動機となる要求。この要求がかなえられないのであれば、システム開発の意味がないというレベルのもの。

「ユーザ要求」: ユーザがシステムで行う作業についての要求。実現しなければユーザの仕事に支障がある。

「機能要求」: 業務要求やユーザ要求をシステムとして実現するために必要な機能に関する具体的な要求。

最初の「業務要求」は要求仕様書の最初の部分などで、「このシステムの目的」いうような標題の下で明確に記述すればよい。その上で要求仕様書の本体の部分では、2 つ目の「ユーザ要求」と 3 つ目の「機能要求」を仕様とペアにして記述する方法がある。

我々が定義する要求仕様書の書き方と、それを採用する理由

既に記したように、我々は清水吉男氏が提案する「USDM（Universal Specification Description Manner）表記法」を我々の要求仕様書の本体部分の書き方として採用することにした[SIM05]。

それを採用する理由は、この方法を使うことによって、結果として良い要求仕様書を作成することができると我々が考えたことにある。具体的には、この方法で以下のことが実現可能である。

1. USDM 表記法では、まずシステムへの要求を明記にする。要求を 2 段階で記述することにし、その下位の要求を引き出すための基準がある。これによって、要求を明確にすることがより容易になる。
2. 次にシステムの仕様を、該当するシステムの要求の下に記述してゆく。こうすることで、システムの有効な仕様を引き出すことができ、仕様を漏れなく、整合性が取れた形で抽出することができるようになる。またこれによって、仕様のレビューもより容易になる。
3. システムへの要求に、なぜその要求をするのかについての「理由」を明記する。この「理由」を書くことによって、その要求を必要とする背景が明らかになり、

仕様のレベルで別のより良い解を得ることも可能になる。

4. 仕様を明確にしている段階でもう一歩進んでそれを実現する具体的な方式まで頭に浮かんだら、それを「説明」として追記することで、要求仕様書に一層の深みを持たせることができる。

なおこの USDM 表記法は、前述の IEEE の規格の中で要求仕様書のひな形として定義されている 8 種類のものとは、大幅に異なっていることをここで明らかにしておきたい。

要求仕様書のプロトタイプ

要求仕様書の要求と仕様を記述する部分には、USDM 表記法を採用すると述べた。しかしいきなりこの要求と仕様を記述する部分に入るのではなく、要求仕様書にも導入部分がある。例えば、秋本氏らの言う「業務要求」などを記述する部分である。

これについては、IEEE の規格に 1 つのプロトタイプが提示されている。それをこの報告書の巻末に「付 2」として記述する[IEE98]。

そして以下で、要求仕様書の中の要求と仕様を記述する部分に採用する「USDM 表記法」について述べる。より詳しくはこの報告書の巻末に付 5 として、清水氏が 2006 年 8 月の我々の研究会でプレゼンテーションをしたときに用意されたスライドを清水氏のご厚意により掲載してあるので、それを参照願いたい。また参考文献にあげた清水氏の著書[SIM05]も、これについての重要な情報源であることは言うまでもない。

「要求」とは何か

以下で、この「USDM 表記法」の概要を述べる。

先ず考えたいのは、「要求」とは何か、ということである。清水氏によれば、要求とは「実現したいことのゴールである」とする。端的に「実現して欲しいこと」といっても良い。例えば「お風呂が沸いたら、すぐにお風呂に誘導したい」は要求である。

機能的要求は、「振る舞い」と表現することもできる。振る舞いとは一般に「イベント」に始まり、「入力処理」、「変換処理」、「出力処理」などの一連の処理を終えて停止するまでの「動き」を意味する。つまり止まっている状態から何らかのきっかけを基に動き出し、一通りの動作を終えて再び停止するまでの範囲を持つ。これが「ゴール」、すなわち「実現して欲しいこと」である。

この「要求」は後で述べるように、情報システムを開発する場合に「仕様」を導き出すたいへん重要なものである。この重要なものがヒアリングの場などで言葉として交わされることはあっても、情報システムの開発に伴って作成される文書類には、一般にどこにも表現されていないという。しかし USDM 表記法では、これを明確に記載する。

また要求は、「範囲」を見せるように表現する必要がある。範囲が広い場合は階層化して、範囲を狭める努力が必要である。この結果として要求は、上位要求と下位要求に分けられる。要求を分割する基準として、次の 4 つがある。

- 時系列分割：時間軸に着目して、要求を分割する。
- 構成分割：「機能」や「構成」で要求を分割する。
- 状態分割：「状態」という概念で要求を階層化する。
- 共通分割：共通する機能を取り出して独立させる。

この要求の役割は、有効な仕様を引き出すことにある。

「理由」を付ける理由

要求には、それが存在する理由がある。その理由を上位要求、下位要求を含む全ての要求について明示する。そうすることによって要求の背後にある理由を探ることができ、仕様を外さずに捉えることができるようになる。

お風呂の件に関わる上記要求の理由には、「燃料の消費を節約する」が該当する。

「仕様」とは何か

それでは「仕様」とは何だろうか。仕様とは「要求（実現して欲しいこと）を満たすための“具体的な振る舞い”の記述」である。したがって仕様は、必然的にいずれかの要求に属することになる。

さらに仕様は、最終的には何らかの形でプログラムのコードに変換されるものであり、「それが実現されていることを検証することができるもの」である。つまり仕様は、実行可能でなければならない。別の言い方をすれば、「仕様」はプログラムを読むことで把握することができる。しかし「要求」は、そのような方法で把握することができない。

上記お風呂の件の要求に対する仕様の 1 つとして、「“沸く”状態の 1 分前に、『もうすぐお風呂が沸きます』と知らせる」が該当する。

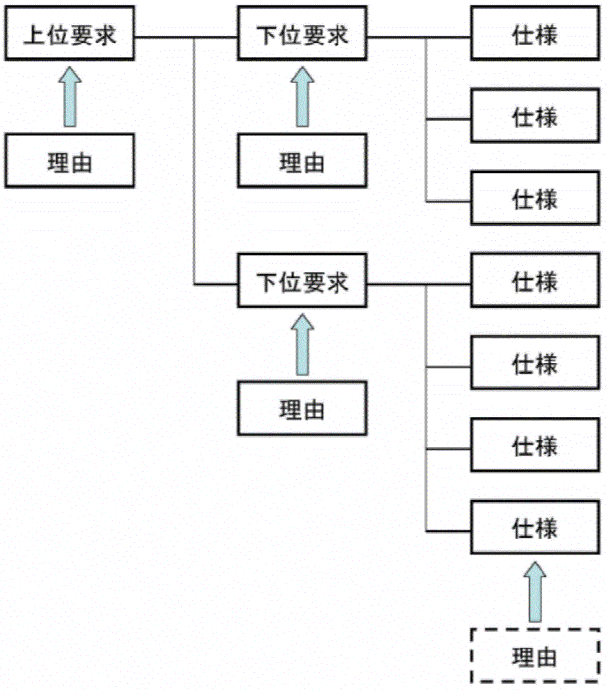
当研究会のメンバー企業の中に、既に USDM 表記法による要求仕様書作成を実施しているところがある。その企業では、「仕様」にも必要に応じて「理由」を書くことを推奨している。ただし、「要求」に対して「理由」を書くことは必須であるけれど、「仕様」について「理由」を書くことは必須とはしていない。理由を書くことでその仕様をについてより良く説明できると思われるときに書くことを推奨するというレベルに留まっている。

「要求」と「仕様」の関係

「要求仕様書」とは、上記要求と仕様を別々に記述したものではなく、要求書と仕様書を合体したものでなければならない。つまり表現された要求の中に、仕様を納める形で記述することが必要である。この要求と仕様の関係を、図表 3-2 で示す。

ここでは要求を、上位要求と下位要求に分けている。USDM 表記法では、要求は常にこのように 2 階層で表す。清水氏は、「ユースケース」が上位要求であれば、「シナリオ」が下位要求に対応するとしている。あるいは秋本氏らが定義した「ユーザ要求」が上位要求、「機能要求」が下位要求と考えることもできる。

このような方法で仕様を明確化することで、的確に仕様を挙げ尽くすことが可能になる。



図表 3-2　要求と仕様の関係

トレーサビリティへの対応

最近のソフトウェア工学では、仕様についてのトレーサビリティの議論が盛んである。つまりどの仕様がどういう形の設計になり、さらにその設計に基づいてプログラムのどの部分でその仕様が実現されているかを問題にしている。これを実現することにより、保守時の対応などで力を発揮することになる。

このトレーサビリティを実現するために、USDM 表記法では要求と仕様に「記番号」を付けることにしている。この記番号が設計書やプログラムの中にまで持ち込まれ、それを使うことで仕様から設計、プログラムまで、一貫して追跡することが可能になる。この件については、さらに「８．要求仕様を設計からプログラムまでトレースする方法」で議論する。

非機能要求の対応

非機能要求とは前述の通り、品質要求や保守性、移植性など、情報システムが持つべき機能以外のものへの要求である。これらの非機能要求にも、USDM 表記法では機能要求と同じ形で表現することができる。

つまりこの場合「要求」欄に記載される要求が「非機能要求」になり、それを実現するための「仕様」が要求とペアになって記載されることになる。これは「機能要求」とそれを

実現するための「仕様」を記載する場合と、何ら変わるところがない。この点が、USDM 表記法の１つの特徴であるということができる。

「説明」の利用法

すぐ後で述べるが、この仕様の部分を適切に記述するためには、プログラムの設計についてのバックグランドが必要である。したがって仕様を記述している人が、それを記述している過程で、その仕様の良い実現方法に気づくことがあるだろう。

しかし仕様を具体的にどのような形で実現するかを決める責任と権限は、後でその情報システムの設計を担当することになる人たちが持っている。仕様を決める段階でそれについて口を出すことは、越権行為となる。しかし仕様を書いている人がその段階で気づいた実現方法は、あるいは本当に良い方法かも知れない。

このような場合に備えて USDM 表記法には、「説明」という項を設けて、そこには具体的な実現方法などを記載しても良いとしている。もちろんここに書くものは、単なる付加情報でも良い。ここに書かれている情報を利用するかどうかは、後で設計を担当する人が決めればよい。しかしこれがあることで、この要求仕様書は一層の深みを増すことになる。

仕様漏れや仕様の衝突などの発見法

清水氏によれば、的確に仕様が記載されていると、SE はそこから仕様漏れや仕様間の矛盾、仕様の衝突などを発見する能力を持っているという。つまり USDM 表記法を使用して要求仕様書を記述すれば、後は特別の方法を講じなくても、仕様の漏れや仕様の衝突は発見可能と、清水氏は言う。

この問題については、さらに「６．要求仕様の定義方法」で議論したい。

要求仕様書を記載するためのツール

清水氏によれば、要求仕様書は Word などのワープロソフトを使って記述するのではなく、Excel を使って記述するのが良いという。Excel を使うことで、要求（上位、下位に分けたもの）、理由、仕様、説明、要求と仕様に付けた記番号、それぞれについての内容などを、的確に記述することができる。

さらに Excel には、ある場合特定の行を折りたたんで表示しないようにする機能が当初から用意されており、その機能を有効に使うことで要求仕様書を要求書として見たり、仕様のグループ化を検討したりすることが可能になる。

清水氏の著書で彼が用意した要求仕様書の本体部分のテンプレートの 1 つを、「付 3-1」としてこの章末に添付する[SIM05]。

要求仕様書は誰が書くか

最後のテーマは、一体要求仕様書を誰が書くべきなのかというものである。

要求部分は、ユーザが書くのが妥当だろう。しかしユーザの立場で仕様まで的確に書くことができるのだろうか。多分難しい。

このことから要求仕様書は誰が書くべきなのかという問題について、三通りの答えが用意されている。その最初のものは「SE が書くべき」とするもの、二つめは「ユーザと SE が共同で書くべき」とするもの、そして最後は「『要求エンジニア』と呼ばれる特別の訓練を受けた人が書くべき」とするものである。

SE だけが書くとする場合その SE は、ユーザから作業中に充分な情報の提供を受けるか、事前にその領域について勉強しておくことが必要である。したがって実際に要求仕様書を記述するのは SE だけかも知れないが、実質は二つ目の「ユーザと SE の共同作業」の形を取っていることになる。

要求エンジニアとは、ユーザ出身でも SE 出身でもかまわないけれど、SE 出身の場合は適用業務の領域に深い造詣を持っていて、ユーザに代わって要求を書けるレベルになることが必要である。逆にユーザ部門出身の場合は、仕様のレベルを的確に書くための特別の訓練を受ける必要がある。いずれにしろ「要求エンジニア」という言葉はまだ一般的ではなく、例えば経済産業省が定めて公表した「IT スキル標準」[IPA06a]にも、この技術者は定義されていない。

清水氏は、「ユーザと SE が共同で書くべき」とする立場をとる。ユーザが要求部分を書き、SE が仕様部分を書くというように、作業を分担する方式である。この場合まずユーザが要求部分を完成させて、それを基に SE が仕様部分を追加してゆくという方法が可能である。複数の SE が分担を分けて仕様部分を書いてゆくことも可能と、清水氏は指摘する。

前にも記したが、仕様を書くためにはプログラムの設計などについてのバックグランドが必要である。我々も素直に、この清水氏の見解を取り入れることにする。

今要求仕様書の作成に、SE が関わらないケースが多いように思える。その場合本来は「仕様」も記載するべき要求仕様書に、本当の意味での「仕様」が書かれていないと清水氏は指摘する。

この場合には、本当の意味での「仕様」は設計作業に入ってからでなければ抽出できないという状況になり、それまでに発見されているべき「仕様の漏れ」、「仕様間の不整合」などが発見されないまま設計作業が行われる。このためこれらの問題がテストの最終段階で明確になって、問題を起こすという状況が生じる。清水氏の主張の背景には、このような認識がある。これが、我々が清水氏の見解を取り入れた理由の 1 つでもある。

この要求仕様書をユーザと SE が共同で書くための作業分担や留意点などを、「7．ユーザ部門と IT 部門の役割」で議論する。

「SE」という言葉は、ここでは「ソフトウェア・エンジニア」を表す言葉として捉える。しかしそれでもこの言葉はたいへん幅が広く、ある意味であいまいな言葉である。SE とは一般に、情報システムの企画や分析といった上流からテストの実施／統括といった下流まで、あるいはソフトウェア・プロセスの改善や標準化の推進、要員教育など、たいへん幅

広く活躍することができる人材を意味する。我々がここで述べている「SE」とは、最低でもプログラム設計ができるスキルを持ち、ユーザ企業の情報システム部門所属の SE をイメージしている。しかしそれには必ずしも拘らず、ユーザ企業の情報システム子会社所属の SE であっても良く、場合によればソフトウェア会社所属の SE であってもかまわない。つまり「要求を仕様に落とすことができる」人であれば、その所属する企業には拘らない。

要求仕様書そのものについての留意事項

IEEE の要求仕様書についての規格では、この要求仕様書について、さらに次のように述べている[IEE98]。

- 要求仕様書は、開発が進むにつれて変化してゆくものである。つまり開発の過程で欠陥が見つかったり、充分な精度がないことが明らかになると、改訂されなければならない。
- 要求仕様書は、その時点で分かっている限り完全に記述されていなければならない。たとえ変更が避けられなくても必要な記述がなされていなければ、それは要求仕様書として不完全であり、そのため不合格になる。
- 要求仕様書の変更は、公式の変更プロセスを適用してなされなければならない。ここでいう「公式の変更プロセス」とは、ソフトウェアの構成管理を意味している。つまり要求仕様書が変更される場合は、ソフトウェアの構成管理のルールに則って変更されなければならない。

我々の要求仕様書について

「５．知識共有の方法」と「６．要求仕様の定義方法」で、我々の要求仕様書はここで述べたものを核にして、これに概念データモデルと入出力の定義を加えたものであることを述べる。また「10．画面／帳表のデザインとユーザビリティ」で、ユーザビリティへの対応方法についても述べる。

最終的にこれが、我々が定義する要求仕様書全体ということになる。

要求仕様書の記載についての補足

現行システムを再構築する場合、新しく追加される機能については要求仕様書にここで述べたような形で、その要求と仕様が一般に記載されているのが普通である。しかし現行システムが持っている機能を新システムに継承する場合、単に「現行機能保証」とだけ記載し、それ以上のことを明記しないケースがある。これは IPA も指摘する通りに絶対に避けなければならないことであることを、ここで明記しておきたい[IPA06b]。

また要求の曖昧さを避ける方法として、「形式仕様技術」がある。形式仕様技術とは数学的な表現を用いて、仕様を誤解がないように、明確に要求を記述する技法である[PRE05]。しかし今の日本で、少なくともビジネス・アプリケーションの要求仕様書の記載では、こ

の技法を用いることができる技術者は少なく、この表現で書かれた仕様を理解できるユーザはもっと少ないと思われる。したがってこの技術については、今後の課題としておきたい。

参考文献

[AKI04] 秋本芳伸、岡田泰子著、「若手 SE のための要求仕様のまとめ方」、(株) ディー・アート、2004 年.

[COC01] アリスター・コーバーン著、ウルシステムズ（株）監訳、「ユースケース実践ガイド-効果的なユースケースの書き方」、（株）翔泳社、2001 年.

[CRO79] フィリップ・B. クロスビー著、小林宏治監訳、「クオリティ・マネジメント：よい品質をタダで手に入れる法」、日本能率協会、1980 年.

[IEE98] IEEE-SA Standards Board, “IEEE Recommended Practice for Software Requirements Specifications IEEE Std 830-1998”, The Institute of Electrical and Electronics Engineers, Inc., 1998.

[IPA06a]「IT スキル標準 v2」、独立行政法人情報処理推進機構 IT スキル標準センター、平成 18 年 4 月 1 日.

なおこの資料は、以下の URL からダウンロード可能である。

http://www.ipa.go.jp/jinzai/itss/download_V2.html

[IPA06b] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター編、「経営者が参画する要求品質の確保～超上流から攻める IT 化の勘どころ～第 2 版」、（株）オーム社、平成 18 年 5 月 25 日、ISBN4-274-50076-4.

[ISO00] 日本工業標準調査会審議、「品質マネジメントシステム－基本及び用語 JIS Q 9000：2000 （ISO 9000：2000)、日本規格協会、平成 12 年 12 月 20 日.

[PRE05] ロジャー・S. プレスマン著、西康晴他監訳、古沢聡子他訳、「実践ソフトウェアエンジニアリング ソフトウェアプロフェッショナルのための基本知識」、日科技連、2005 年.

[SAN94] J. サンダース、E. カラン著、原田曄他訳、「ソフトウェア品質向上のすすめ－新しいソフトウェア開発の標準」、(株) トッパン、1996 年.

[SIM05] 清水吉男著、「[入門+実践]要求を仕様化する技術表現する技術～仕様が書けていますか」、(株) 技術評論社、平成 17 年.

[SOM97] Ian Sommerville、Pete Sawyer 著、富野壽監訳、「要求定義工学プラクティスガイド」、(株) 構造計画研究所、2000 年.

付 3-1. 要求仕様書（要求と仕様記述部分）のテンプレート

					備考欄
カテゴリ名 （記号）	要求	（要求番号）	ここに要求を記述する。		
		理由	要求の背景や理由について記述する		
		説明	要求について必要に応じて説明する。仕様とは見なされない。セルを広げて図を貼り付けることも可能。		
		要求	（要求番号）	ここに階層化された要求を記述する。要求番号は一段下げられる。	
			理由	範囲を狭めた要求について、背景や理由を記述する。	
			説明	要求について必要に応じて説明する。仕様とは見なされない。	
		<<仕様分類名>>		主分割記号・・・全体を通して共通の分割基準として決める。	
			<<仕様分類名>>	補助分割名・・・主分割の中に異なるテーマの仕様が混在する場合、補助分割記号を使って純度の高い集合を作る。	
		□	（仕様番号）		
		□	（仕様番号）		
			<<仕様分類名>>		
		□	（仕様番号）		
		□	（仕様番号）		
		□	（仕様番号）		
		□	（仕様番号）		

４．要求仕様書作成に関わる作業

要求仕様書作成に関連する作業

要求仕様書は今対象としている情報システムの構築で、設計に先立って作成される文書である。このことは、「３．要求仕様書とは」で既に述べた。従って、当然その情報システム構築に伴って作成される他の文書と、大いに関連を持つ。例えば設計段階で作成する各種の設計書は、この要求仕様書に書かれた要求をいかに実現するかについて明記した文書類である。

しかし単に後方向だけではなく、要求仕様書はこの前に実施される IT 戦略と充分な関連を持つべきである。つまり IT 戦略で決めた枠組みの中で、今対象としている情報システムの機能や位置づけ、規模などを明確にしてゆかなければならない。これが、IT ガバナンスの基本の１つである。さらに、要求の膨張を防止する手段の１つともなる。

この章では、特に IT 戦略との関わりにポイントを置いて、この要求仕様書作成作業に関連する他の作業に何があり、それが要求仕様書作成作業とどう関連するかについて考えてみたい。

要求の膨張に関わる問題

要求仕様書作成に関わる問題には、いろいろのものを挙げることができる。それらは既に「２．要求仕様書に関わる問題とその解決方法」で問題提起をし、いくつかのものについてはそこで解決方法についての議論をした。

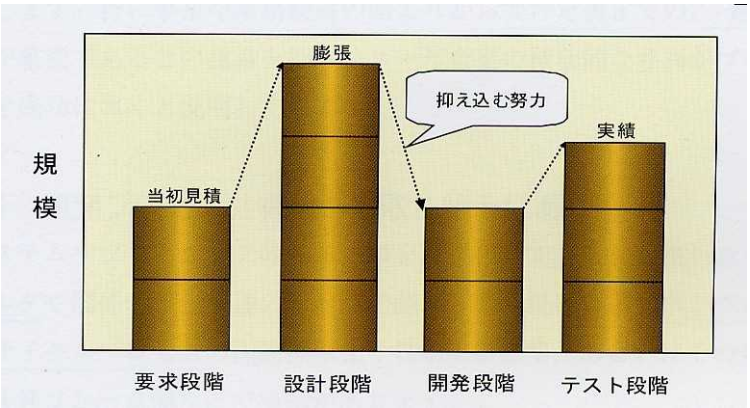
ここではその中の、特に要求の膨張問題を取り上げたい。要求の膨張とは、要求定義の段階で明記された要求が、開発作業が進むにつれて増大することを意味する。常に規模が増大するだけではない。増大した規模を元の小さなものに戻そうとする努力も当然行われる。しかし仮にそのような努力があったとしても、これらを総合した結果として要求の規模が増大してしまうという問題は、一般に避けることが難しい。

要求の膨張を、何故避けなければならないのか。ある情報システムへの要求が膨張すれば、当然開発予算が超過する。スケジュールも当初の要求で見積もったものから遅れるだろう。これらは IT ガバナンスについての能力不足を露呈したものと考えることができ、当然避けなければならないものである。

しかしそれ以上に大きな問題は、予算も含めた広い意味でのソフトウェアの開発能力をある特定の情報システムに重点配分してしまい、その結果当初の IT 戦略立案時に実現したいとした他の情報システムの実現が難しくなることにある。IT 戦略は基本的に、企業の経営戦略をベースにして策定されている。IT 戦略実現に関わる問題は、経営戦略実現に支障が生じる可能性がある。このことは単に、IT ガバナンスの能力不足を露呈する程度の問題ではない、もっと大きな問題を引き起こすことになる。

情報処理推進機構（以下、IPA）のソフトウェア・エンジニアリング・センター（以下、

SEC）では、SE の経験則として「2-4-2-3 の法則」をあげている（図表 4-1 参照）。



図表 4-1　2-4-2-3 の法則（[IPA06b]より）

「2-4-2-3 の法則」とは、次のようなものである。当初の要求定義の段階でこの要求の規模を 2 と見積もる。そして開発に関わる予算や開発要員などを、2 の規模に対応するだけ用意する。しかし次の設計段階で新たにいろんな要求が加わり、その規模は 4 まで増大する。この 4 のままでは予算超過やワークロード増加によるスケジュール遅れが必至であるから、開発を開始する段階でこれを元の 2 にまで縮小する。しかしさらに揺り戻しがあり、最終的に規模は 3 で決着する、というものである[IPA06b]。この結果として、ユーザ側から見ると機能が 1 足りず、開発者側から見るとコストが 1 多くなり、結果として双方に不満が残ることになる。

実際の規模とその推移が 2-4-2-3 になるかどうかはともかくとして、これはよく見られる現象ということができる。

要求の膨張を防ぐ手立て

この要求の膨張が起きる原因は何か。これが起きたことで、結果として次にどういう問題が起きるのか。この要求の膨張を防ぐ手立てはあるのか。次にこういうテーマについて、考えてみたい。

要求の膨張は、なぜ起きるのか。SEC はこの原因として、ユーザ部門（現業部門）の「やりたいこと」と開発部門の「できること」の間だけで綱引きが行われ、結果としてユーザ部門が力で開発部門を押さえ込んでしまうことに原因があると分析している[IPA06b]。あるいは別の見方をすれば、開発部門の人はできるだけ多くのユーザ部門の要求を実現することでユーザ部門の人に喜んでもらいたい、そのためには開発の苦労もいとわないという、単純な好意に満ちあふれている。

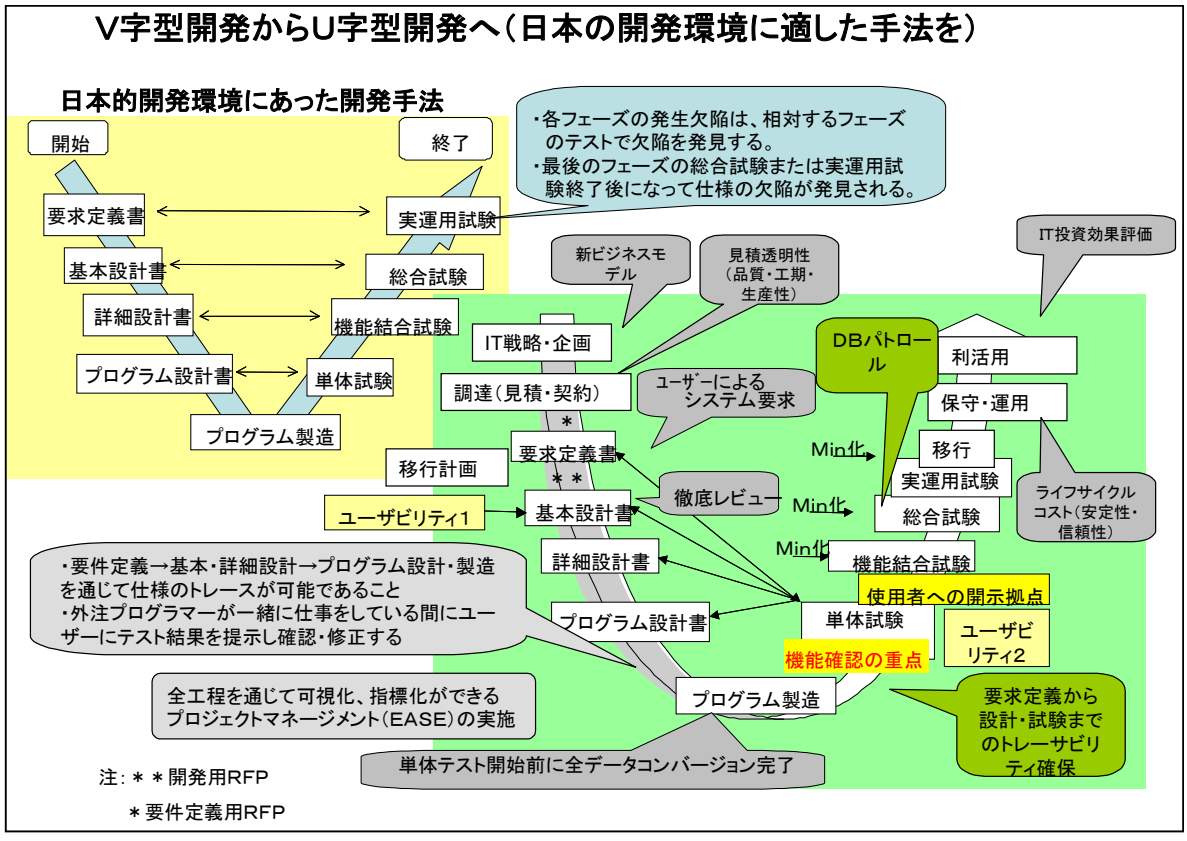
この結果として、次のどういう問題が起きるのか。ユーザ部門にも開発部門にも結果として不満が残ることを既に指摘した。それ以外に、個別最適は実現できるかも知れないけれ

ど、全体最適が実現されないという事実を指摘したい。全体最適を実現するためには常に企業全体を視野に入れ、その全体についての将来の姿、進むべき方向をまず明確にした後で、その全体の中での個々の役割、位置づけを意識し、要求などを明確にしてゆかなければならない。ユーザ部門と開発部門だけが関係する要求定義では、この全体最適を実現するというスタンスは実現しにくい。

全体最適実現の方法

それでは、どうすればこの全体最適を実現することができるのだろうか。日本情報システム・ユーザー協会（以下、JUAS）が提案する「U 字型開発モデル」（図表 4-2）では、要求定義書作成のフェーズの前に「IT 戦略・企画」のフェーズを置いている。

「IT 戦略」とは、企業戦略の根幹をなす「経営戦略」に基づいて、その経営戦略で明確に示された目標を達成するために IT 部門全体として何を、どうしなければならないのかを定めたハイレベルのもの、ということができる。ここで、個々の情報システムについて議論する必要はない。議論の対象は、あくまで「全体」である。



図表 4-2　U 字型開発モデル（[JUA05] より）

個々の情報システムについての議論は、次の「IT 企画」の段階で行われる。ここではそ

れぞれのシステムについて、そのシステムが対象とする範囲／領域、システム化の目的、実現する効果、開発／運用の費用の見積もり、その情報システムを入手するための手段（自社開発か、パッケージの購入か、あるいは ASP（アプリケーション・サービス・プロバイダ）が提供するサービスを受けることで済ませるか）などを明確にして、その情報システムを実現するか否かを決定するための判断資料を作成する。

企画作業の結果に基づいて、常に情報システムの開発／購入が決裁されるわけではない。「この情報システムを開発する必要はない」との決裁が行われることもあり得る。しかしここで「開発」についての決裁が行われると、要求定義から始まる一連の開発工程がスタートすることになる。

この「IT 企画」の作業を、ここでは「ビジネス検討」と呼ぶことにする。常にそうだというわけでは必ずしもないけれど、最近は新しいシステムの開発が新しいビジネスのスタート、あるいは従来からのビジネスの方法などの変更を契機に行われることが多いという潮流をふまえての命名である。もちろんビジネスの方法などは何も変えず、今使っている情報システムの老朽化が進んだので、単純に再構築しようというケースもない訳ではない。しかし最近は新しいビジネスを始める場合などに、情報システムを準備することが不可欠の要件になりつつあることは事実である。

この「ビジネス検討」では、IT 戦略で明確にした情報システムの全体像と全体の中での個々の情報システムの位置づけをふまえて、今対象とした情報システムについて詳細な検討を行うことが必要である[IPA05]。

ビジネス検討

繰り返しになるが、この「ビジネス検討」の前にすでに戦略の立案と承認が終わっていることが大前提である。戦略とは、「個々の企画（ここの言葉では「ビジネス検討」）の前に行う、全体としての方向決め」である。ビジネス検討の段階では、IT 戦略よりもっと基本的な経営戦略や営業戦略が、情報システムの方向性を決める物差しとして使われる。

ビジネス検討では、「この全体としての決められた方向の中で、今テーマになっているビジネスの実施は整合性がとれているのか」についての検討から始める。つまりこの結論を得るために、ビジネスで何を実現したいのかを明確にすることがユーザに求められる。

ビジネスを開始することについての方向性が示されれば、そのビジネスの遂行を支援するために必要な情報システムに議論が移る。この議論もユーザの視点を基に、ユーザ主導で行われることが必要である。

ここでは、この情報システムについての詳細を明らかにする。具体的には、以下の事項を明確にすることになる。ここで取り決めた内容が、IT 戦略と整合性をとっている必要があることは言うまでもない。

- その情報システムを必要としている理由
- その情報システムの目的

- システム完成時のその情報システムの、全体における位置づけ
- その情報システムに必要な機能
- その情報システムを用意することによって得られる効果
- その情報システムを用意するために必要な費用
- 上記 2 項を基にした効果分析（投資回収期間の計算など）
- その情報システムを用意する方法（自社開発か、パッケージ購入か、など）
- サービス開始時期
- 品質目標
- QCD（品質、費用、サービス開始時期）の優先順位付け

この結果に基づいて、その情報システムの開発／購入を実施するのかが決裁される。

ここで、この情報システムの「機能」について述べておきたい。ここで取り決めた「機能」が、前述した将来の機能の膨張を防ぐための防波堤の役割を果たすことになるからである。

この意味から、機能はできるだけ具体的に、明確に取り決めることが必要である。つまり、一部この次の作業である要求定義や、さらにその先の作業である外部設計の作業で行う情報システムからの出力の定義、処理の方式などを先取りして、ここで取り決める必要がある。またここで詳細なレベルまで検討しておくと、もう少し先で行う見積もりの誤差を小さくできるという効果も期待できる。

さらにこの段階で何かを決定した場合、議事録にはその結論を記載するだけでなく、なぜその結論に至ったのかを明示するための検討過程などの記述も欠かせない。要求定義以降の作業で、作業の揺り戻しや手戻りを防ぐために必要となることが多いからである[IPA05]。

設計

要求定義書の作成とそれに続くシステム分析の作業で、その情報システムは、機能だけでなく非機能と呼ばれるものも含めて、何を実現するべきか、つまり“What”が明らかにされる。設計はこの情報システムが具備すべきとされた機能と非機能を、いかにして現実のネットワークとコンピュータの上に実現するのかということ、つまり“How”を明らかにする作業である。

この作業の進め方の詳細を、ここで議論するつもりはない。ここでは、要求の膨張をもたらさない方法だけを述べておきたい。

要求定義書作成の段階でも機能の膨張は起きる。しかし要求定義書作成はビジネス検討の実施からさほど時間がたっていないため、そこで取り決めたことについての記憶がまだ新しい。設計は逆にビジネス検討から少し時間がたってから実施されることになる上、システムの機能などが順次目に見える形になってくるためユーザ部門から新たな要求が出やすく、要求や仕様の膨張が起きやすいフェーズということができる。

つまり設計作業で決められたことが、そのまま情報システムに反映される。逆の言い方をすると、ここで落とされたものが情報システムに取り込まれることはない。その事実を、

ユーザを含めて全員が認識している。このことが、この段階で要求や仕様の膨張をもたらす原因になる。つまりユーザが「ある機能がぜひ必要」と考えると、その機能を盛り込むべくこの段階で精一杯の努力をすることになる。

さらに設計作業では「仕様」を具体的に「処理の方法」に落としてゆくため、要求仕様書ではハイレベルの状態でとどまっている要求や仕様が、より具体的な「シナリオ」や「振る舞い」のレベルで認識されることになる。このため要求仕様書には記載されていなかった別の「仕様」や、時には「要求」が引き出されることがある。これらがさらに、要求や仕様の膨張をもたらすことになる。

この要求や仕様の膨張を防ぐためには、必要に応じて「システム検討」の結果に戻ることしか方法がない。しかし「システム検討」では一般に、さほど具体的なことが取り決められているわけではないから、その「精神」に立ち戻り、全体最適の観点から必要なものと必要ではないものを区分けして、必要なものだけを取り込んでゆくスタンスが重要である。別の言葉を使えば、ビジネス検討の作業実施に当たって、こういう重要な作業を行っているのだという自覚が欠かせないということになる。

それでも仕様が拡大し、いかにして機能を落とすかを考えなければならなくなった場合には、機能に優先度を付けて対応するのが望ましい。その手法が、JUAS が作成した「システム・リファレンス・マニュアル（SRM）に記述されている[JUA05]。

なお我々は、我々が「ビジネス検討」と名付けたシステム企画の段階で、普通なら外部設計で行うことになっている情報システムの出力の定義などを行うこととした。この後の第 6 章でさらに、要求仕様書を作成する段階で入出力の定義を行うことを我々は提案する。つまりこれまで設計段階で行うとされていた作業のいくつかを、もっと前倒しで行うことを我々は提案している。ビジネス検討の段階でこのように対応することによって、機能をより一層明確にできると指摘した。要求仕様書作成段階でもこうすることによって、「仕様」をより一層明確にできるという効果がある。ただし設計作業では画面のレイアウトの決定など、まだ行うべき多くの作業があることを明記しておきたい。

プロジェクト評価

最近ビジネス界などで、「PDCA を回す」という表現がよく使われる。あることを実施するに当たって、まず計画（Plan : P）を立て、次にその計画に基づいて実行（Do : D）する。そしてその実行結果をチェック（Check : C）し、何か問題などがあればそれを取り除くのは当然のこととして、なぜその問題が生じたかの原因を究明し、同じ状況が起きても再びその問題が起きないようにするにはどうするべきかを考えて作業のやり方などを修正（Action : A）する、というわけである。P も D も C ももちろん大切だが、ある見方をすれば最も大切なものは A なのかも知れない。

この PDCA サイクルは、普通は何回も、何重にも回される。例えばソフトウェア開発では、情報システムの分析段階で一度 PDCA サイクルを回し、設計段階で別の PDCA サイク

ルを回す。プログラミングの段階、テストの段階も同様にする。さらにその上で、これらの作業をすべて包含するプロジェクト全体で、大きく PDCA サイクルを回す。

このプロジェクト全体で大きく PDCA サイクルを回すとき、C に該当する作業がプロジェクト評価である。つまりこのプロジェクト評価が対象とするものは、要求定義書作成だけではない。我々が「ビジネス検討」と呼んだシステムの企画作業から始まり、システム構築などのプロセスを経て最終的に情報システムを稼働させるまでの一連のプロセス全体が、ここでの C の対象になる。

ここではその中で、要求定義書作成との関連に絞ってプロジェクト評価を見てみたい。

プロジェクト評価は情報システムがカットオーバーして、半年から 1 年後ぐらいに行われるのが普通である。カットオーバー直後はまだ実運用の経験や時間が充分ではなく、当初想定した効果が実現されているのかどうかの判定が難しい。この判定ができる状態になって、プロジェクト評価を実施する必要がある。

プロジェクト評価の内容は多岐にわたるが、要約するとその情報システムが当初想定した効果を上げているかどうかを判定することにつきる。仮に何か問題があるとすれば、前述の通りその問題の根源を洗い出し、取り除いたり修正したりして、次のプロジェクト以降で同じ問題の発生を抑制するための行動に結びつけるきっかけを作る。

あるいはプロジェクト評価の対象をもっと大きくして、その情報システム構築などのきっかけとなったビジネスの実施そのものをチェックの対象にするべきなのかも知れない。

いずれにせよそのような大きな枠組みの中で、要求定義書作成に関わるチェック項目は以下のようなものとなる。

- 当初計画の妥当性評価
- 稼働システムの効果把握
- 全体のシステム像確認
- ユーザの満足度把握
- ユーザビリティの評価
- 投資対効果の把握（実績に基づく ROI、KPI 等の確定を含む）
- 品質の実現値の把握／実現度の評価
- 性能目標実現の確認
- 開発プロジェクトの活動の評価
- 運用状況の評価

これらの評価の結果何か問題があれば、その問題の根源を取りのぞき、次回のプロジェクトのビジネス検討や要求定義書作成で、同じ間違いを繰り返さないようにすることが肝要である。

この章の議論のまとめ

この章のここまでの議論のまとめを、この章末に「付 4-1 要求定義を中心とした作業の

まとめ」として添付する。

ビジネスプロセスの改善

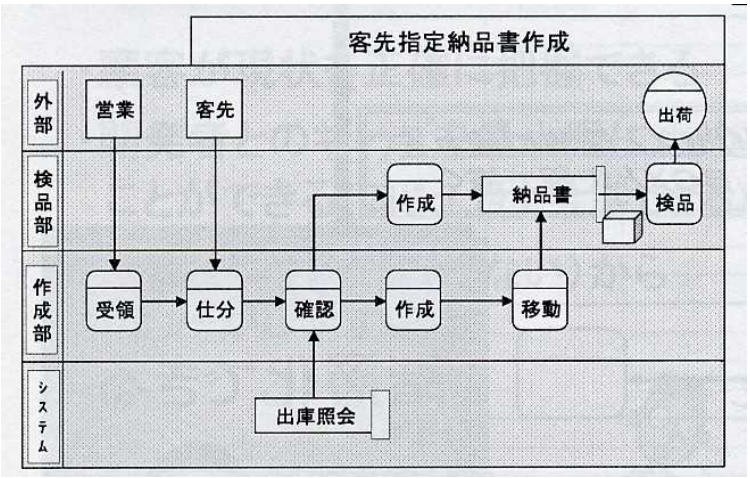
IT 企画やビジネス検討とは直接の関わりを持たないが、要求仕様書作成時に行うべき作業の 1 つに、ビジネスプロセスの改革・改善の作業がある。

ビジネスプロセスとは、ビジネスに関わる作業のやり方、手順などを意味する言葉である。さらにビジネスプロセスの改革とは、これまでの作業のやり方の抜本的再構築を図るものであり、改善とはこれまでの作業のやり方を見直して、新しいシステムの稼働開始などを契機に無駄をなくし、仕事を迅速化し、結果としてより良い方向に仕事の手順、流れなどを変更しようとするものである。

この研究会の会員企業の 1 つに、PFD（Process Flow Diagram）と呼ぶダイアグラムを考案し、それを用いてこのビジネスプロセスの改革・改善を行って成果を上げている企業がある。

PFD は DFD（Data Flow Diagram）同様、現物理モデル、現論理モデル、新論理モデル、新物理モデルの 4 種類のダイアグラムから構成されるが、ユーザとの共通理解が図りやすく要求仕様の漏れをさけるため時間軸や組織軸等々の工夫がなされている。作業の流れについての現時点の状況をそのまま現物理モデルに表し、組織の区分けなどを取り除いて現時点での重複・無駄を排除して、作業の流れの本質を現論理モデルの形で記載する。それをベースに新たなビジネス要求を加え必要な改革・改善を行って新論理モデルを作る。さらにこれを基に新システム稼働時点の組織等の物理的制約を織り込んで新物理モデルを作るという手順で、ビジネスプロセス改革・改善の作業が行われる。

PFD の例を、図表 4-3 に示す。余談だが、この図（現物理モデル）では検品部と作成部で「納品書作成」の作業が重複している。PFD 作成の手順は既に上で述べたが、「新論理モデル」を作成する時点でこのような無駄な作業を発見し、省いてゆくことになる。



図表 4-3　PFD の例（現物理モデル）

この企業では企業全体のビジネスプロセスを PFD で表し、その作業を支援する情報システムのデータの流れを DFD で表現するという形で、2 つのダイアグラムを使い分け、組み合わせて効果を上げている。

テストへの対応

本論とはまた直接の関係を持たないけれど、U 字型開発モデルに関連して、テストの実施で大きな効果を上げている当研究会への参加企業があるので、その対応方法をここでご紹介したい。

通常の開発モデル（ここではこれを「V 字型開発モデル」と呼んでおく）では、単体テストはプログラム設計で確定された内容を確認し、結合テストでは詳細設計の内容を確認しというように、テストでは分析と設計の作業を逆に遡って順次確認を取ってゆく。したがってこの場合、機能の確認はシステムテストで行うということになる。

一方の U 字型開発では、単体テストの前に全データのコンバージョンを完了させ、そのデータを使って単体テストから機能の確認を行うという考え方を取っている。

紹介しようとする会員企業は、単体テスト段階でソフトウェア会社からテストを終了したプログラム（モジュール）の提供を受け、情報子会社の SE が中心となって「単体確認テスト」と名付けているテストを実施する。単体確認テストは、テストの環境は単体テストと同じものを使用するが、視点がすでに機能面の確認に移っており、テスト期間中に確認するべき全機能の中、入出力の要件を中心に 50%に及ぶ機能要件の確認がこの段階で可能という。さらにこの方式のテストを、結合テスト段階でも継続して実施している。これによって、システムテスト開始の段階で、すでに 75%の機能の確認が終わっているという。

機能の確認を前倒しで行い、問題を早く解決するというスタンスは、U 字型開発の考え方と共通するものである。これによって同社の場合、あるプロジェクトではシステム・テストで発見して解消した不具合はテスト期間中に発見した不具合のわずか 1 %になり、カット・オーバ直前でも SE の月間の残業時間は 15 時間以下に抑えられているという。

さらに同社の場合、「DB パトロール」と呼ぶ DB のデータ項目間の整合性をチェックするツールを用意し、テスト期間中はもとより、実稼働開始後も常時そのツールを稼働させて、問題の早期発見に努めている。

例えば「売上げ」というイベントがあれば、情報システムの中ではこれをお金の流れと物の流れに分けて別々に処理を行い、最終的にその処理結果をそれぞれデータベースに格納している。このデータベースの中の数値は当然一致しているべきであるが、プログラム・ミスを含む何らかのトラブルがあると、この数値が一致しなくなる。「DB パトロール」とは常時この数値が一致しているかどうかの確認を行い、不一致があると即座に報告するというツールである。前述の通り、問題の早期発見にたいへん大きな効果がある。

参考文献

[IPA05] 「プロジェクト編成会資料 XX 情報誌リニューアルプロジェクト」、IPA、2005 年.

この資料は独立行政法人情報処理推進機構（IPA）のソフトウェア・エンジニアリング・センター（SEC）にある「事例検索システム」のページ（URL は https://sec.ipa.go.jp/enterprise/index.php?type=s&ccd=n）に、提供社名は「B 社」、資料名は「プロジェクト編成会資料」として登録されており、そこからダウンロードすることができる。(実際にダウンロードするためには、IPA にユーザ登録する必要がある。)

[IPA06b] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター編、「経営者が参画する要求品質の確保～超上流から攻める IT 化の勘どころ～第 2 版」、（株）オーム社、平成 18 年 5 月 25 日、ISBN4-274-50076-4.

[JUA05] 日本情報システム・ユーザー協会編、「システム・リファレンス・マニュアル（SRM）」、（社）日本情報システム・ユーザー協会、2005 年 9 月.

付 4-1　要求定義を中心とした作業のまとめ

	項目	ビジネス検討	要求定義	設計/製造	プロジェクト評価
ｼｽﾃﾑ化方針	決定／評価するべきもの	情報システム開発の着手決断	情報システムが果たすべき要求(機能／非機能)の	情報システムの機能・非機能要求の設計・実装	情報システムの効果
	プロジェクトの背景、必要性、目的、目標	事業戦略に基づく当該情報システムの必要性／重要性、全体のシステム像の中での位置づけの確認	システム開発のための主要要件確定 システム設計のための主要条件確定	システム構築のための要件確定 システム検証のための要件確定	当初計画の妥当性評価 稼働システムの効果把握 全体のシステム像確認
	移行方法	業務移行の概念案の作成	移行方式案の決定と負荷の予測	移行準備	
ｼｽﾃﾑ化範囲	現状業務フローと課題	現状業務フローに基づく課題認識 解決策の発案	詳細な解決策の立案	――	(当初計画の妥当性評価に含む)
	新業務フロー	業務の変更内容の検討 新業務フローの策定	新業務プロセスの詳細仕様の確定 変更内容の明確化 業務量の予測	プログラム作成上の詳細仕様の確定	(当初計画の妥当性評価に含む)
	着手サブシステムの優先順位	概要案の作成	工期、システム内容等を基に優先順位付け	同左＋変化分の修正	(ユーザ満足度の把握に含む)
ｼｽﾃﾑ要件	他システムとの関係	関連システムとのインタフェースの概要決定	交換するべき情報の内容、タイミングの確認	関連システムとのインタフェースの詳細決定	(全体のシステム像確認に含む)
	画面／帳表	既存システムの画面／帳表の整理と新業務プロセスとの関係整理	基本画面／帳表の仕様提示(基本的内容、使い方)	全画面／帳表等の入出力仕様確定	(ユーザ満足度の把握に含む)
	ユーザビリティ	ユーザビリティ確保方針の決定	重要なユーザビリティ確保対策の決定	(実装設計)	(ユーザ満足度の把握に含む)
	サービスレベルの想定 (ユーザ数、データ量、性能目標、など)	正常時／ピークのレベルの確認、障害停止時の対応、災害時の代替手段等の検討	ピーク時の対策明確化、性能目標、障害停止時間、リカバリー方式、運用管理方式等の決定	―――― (実装設計)	運用状況の評価 性能目標実現の確認
	データベース	業務プロセス及びデータフローの整理	主要データの整理 (ｲﾝﾌﾟｯﾄ/ｱｳﾄﾌﾟｯﾄ情報、キー情報等) 概念データベースの決定	データベースの物理・論理設計 (テーブル設計、格納構造等の設計)	(全体のシステム像確認に含む)
	インフラ構成	ハードウェア・ネットワーク構成の検討、既存インフラとの整合性	システム要件を実現するハードウェアネットワークの基本構成、導入ソフトウェアの決定、概略予算策定	―――― (運用管理等の非機能要件を含めた詳細実装設計)	(運用状況の評価に含む)
ﾌﾟﾛｼﾞｪｸﾄ実行計画	ﾌﾟﾛｼﾞｪｸﾄのQCD目標	QCDのバランスや整合性の確認 QCDの目標値検討	QCDの具体的項目と目標値の設定	レビューの実施 テスト計画立案	品質の実現値の把握 品質の実現値の評価
	プロジェクトの推進体制	実行責任者を含む責任者、ステークホルダーの決定／確認 リーダ決定	全開発関係者、責任者の役割と実行タイミングの確認	開発(レビューを含む)の計画／実施／評価 検証の計画／実施／評価	開発プロジェクトの活動の評価
	開発計画 (要員計画及び全体工程)	要件定義フェーズの要員と予算確保 投入要員数の関係整理	全工程の要員数と質の確保対策の実行案策定 マスタースケジュール	体制確立と開発／検証の推進 詳細スケジュール	(開発プロジェクトの活動の評価に含む)
	リスク分析	リスクの整理と対策検討(工程別リスクの確認)	リスクの明確化及び対応ルールの作成(撤退、縮小等の回避策)	リスクコントロールの実践	(開発プロジェクトの活動の評価に含む)
投資効果	効果	定量効果案策定(定量化が困難なものは定性効果も可)	詳細な定量効果算出 実行上の課題と解決策の提示	――――	定量効果把握(見込みを含む。定量化が困難なものは定性効果も可)
	予算金額／実績金額	開発と運用・保守の概算予算作成	開発の実行予算作成	変更管理予算の確定	開発に関わる総投資／費用把握 運用・保守のための年間費用把握(見込みを含む)
	投資対効果	投資対効果比率の算出 目標のROI、KPI等確定	開発に関する投資対効果比率算出の修正	同左+必要であれば変化分の修正	投資対効果(実績)の把握 実績ROI、KPI等確定 当初計画との差異分析

５．知識共有の方法

知識の共有は何故必要か

要求仕様書に仕様もその背景にある要求も、さらにその要求が出てきた理由も、全て明確に書かれたとして、それで必要とする情報システムを、ユーザが必要とする機能、性能、品質などを網羅して作ることができるのだろうか。答えは残念ながら、「否」である。

ユーザが必要とする機能、性能、品質などを情報システムの上に実現するためには、要求仕様書を作成した側、つまりユーザと、情報システムを構築する側、つまり SE との間に、要求仕様書上に書かれた「言葉」と「暗黙のルール」について、共通の理解と認識がなければならない。

ところが不幸なことに、それぞれの業界には業界特有の言葉がある。それらの言葉は、その業界にいる人にはたいへん便利な、使い勝手の良いものではあるけれど、その業界の外側の人たちには全くなじみがないという場合が多い。その業界の外側の人たちの中に情報システムを構築する立場の SE が含まれる場合、ここから難しい問題が発生することになる。

その言葉を SE が全く知らなくて、素直に質問してくれる場合はまだ良い。SE が本当に理解してくれたということが分かるまで、ユーザは SE にその言葉について説明する機会を持つことができる。しかし仮に、SE 自身がその言葉を理解していると考えていると、質問をしない。しかしその理解がユーザが期待するものとは全く異なっていることがあると、事態は深刻である。

この問題を回避するために、どうすればよいのか。この章では、この問題について考えてみたい。

知識共有のレベル

要求仕様書に書かれた内容の理解にそご（齟齬）を来さないために、ユーザと SE との間で知識を共有していることが必要であることは既に述べた。しかし単に、この両者が同じ知識を共有しているだけでは、まだ充分ではない。込み入った話になって恐縮だが、もう一つ必要なことがある。それは、「『両者がお互いに必要な知識を共有している』という事実を、さらに両者が共有している」ことである。

繰り返しになるが、この問題を整理すると以下のようになる。

「『両者が必要な知識を共有している』という事実を、さらに共有している」場合、双方はこの問題について何ら気にする必要はなくなる。つまり両者は、お互いが共に正しい認識と理解を持っていることを知っているわけであるから、非常に好ましい状態が存在していることになる。最終的に、全ての知識についてこの状態を目指したい。

知識は共有されているが、その事実が共有されていない場合、両者に不安や疑心暗鬼が生じる可能性がある。結果的にはうまくゆく可能性が高いが、無駄や摩擦が発生するかも知れない。あるいは、余計な神経を使わなければならないことがあるかもしれない。

知識が共有されていない場合、そのこと自体がまず問題である。しかし知識が共有されていないということを少なくとも片方が知っている場合は、対応の方法がある。その問題を解決しようとする力が働くきっかけを作り出せるからである。

共有されていないことを両方とも気付いていない場合が、一番大きな問題を引き起こす。

知識共有のために必要なこと（その１）

ユーザと SE が同じ知識を共有するために、IEEE の要求仕様書についての規格[IEE98]には「1.3. 専門用語、頭文字語、略語の定義」という部分があり、そこには要求仕様書の書き手、つまりユーザの立場から見て、「読み手、つまり SE が知るべきと考える言葉」について記述することになっている。これは一歩前進である。なお IEEE の規格のプロトタイプは、巻末の「付 2. 要求仕様のプロトタイプ」を参照頂きたい。

しかしここには、何を書くべきなのかという問題がまだ残っている。頭文字語はまだ良い。略語も分かる。しかし「何が専門用語なのか」が明確ではない。

一般に我々は、「自分が何を知っているのか」、あるいは逆に「何を知らないのか」を知らない。まして自分ではない他の人が「何を知っているのか」、あるいは「何を知らないのか」は全く分からない。このためここに何を書けば良いのかは、一般に明らかではない。つまりここで専門用語として扱わなければならない言葉には、明確な基準が必要になる。

ここではその基準として、「通常の国語辞典（例えば、「広辞苑」の最新版）には出ていない言葉、あるいは出ていても意味が違う言葉」を「専門用語」として、要求仕様書の「1.3. 専門用語、頭文字語、略語の定義」の部分に記述することを提案する。

知識共有のために必要なこと（その２）

仮にユーザが「1.3. 専門用語、頭文字語、略語の定義」を丁重に書いたとして、それで果たして充分だろうか。要求仕様書を作成したユーザ側は、これで SE の知識は充分になると期待するかも知れない。しかし SE の知識がまだ充分ではなく、しかも双方にその認識がなければ、果たしてどうなるか。前述の通り、この場合は問題を起こす可能性が高い。

この場合の対処の方法をこの章末で分析するが、その結論を先に書けば次のようになる。

それにかける時間やお金、手間などを考えなければ、最も望ましい方法は「試しに作って、それを双方で確認する」ということである。これによって双方の理解の不備／不一致は原則として全て明確になり、したがってこれらを除去し、理解と認識を一致させることができるようになる。フレデリック（フレッド）・ブルックスが、「最初に作った情報システムは全て放棄して、もう一度始めから作り直すのがよい」と述べている[BRO75]のは、他の事実もさることながら、この事実も彼は言いたかったと推察する。

しかし現実は、これはたいへんに困難である。我々はこの問題に、そこまでお金や時間、手間をかけることは許されない。したがって、そこまでお金などをかけずに、しかし同じような効果をもたらす方法を我々は見つけ出さなければならない。

その方法として、我々はここで１つ提案をしたい。それは我々が要求定義書を作成した領域について、概念データモデルを作成することである。概念データモデルは今、「実体関連図（ER 図)」あるいは「クラス図」で表されている。我々は、実態関連図を概念データモデルの表記法として採用することにする。

この実体関連図とは、我々がこれから情報システムを構築しようとする領域で、「大切だと考えているもの」、「興味を持っているもの」、あるいは「それについての情報が必要と考えているもの」として何があり、それらがお互いにどんな関係にあるのかを明確にしたものである。つまり実体関連図を通して、我々がこれから情報システムを構築しようとする領域で、データ（実体と関連）のレベルで、ユーザと SE が知識を共有することが可能になる。

さらにこの概念データモデルに表れる言葉、及び処理の内容を記述した言葉などで「専門用語」に当たるものを、要求仕様書の「1.3. 専門用語、頭文字語、略語の定義」に格納することで、言葉のレベルでの知識の共有を実現することができる。

したがって我々が提案する要求仕様書とは、「３．要求仕様書とは」で述べたものを核に、それにまず実体関連図を付加したものということにしたい。

ユーザの立場から

知識の共有というテーマからは少し離れることになるが、ユーザの立場から要求仕様書を書く上で、いくつかの要望がある。

具体的には、以下の通りである

- 通常の処理（「新規」）だけを書けばよいことにしたい。つまり、「取消」、「追加」、「訂正」などについての処理の詳細は、書かなくても良いことにしたい。
- IT 関連の専門の設計用語などを使わずに書きたい。IT 関連の専門の設計用語には、以下のようなものがある。
  - 「全角」、「半角」
  - 罫線の太さ
  - フォントの種類
  - フォントサイズ
  - ……

訂正処理について

要求仕様書に通常の処理、つまり「新規」の処理を書くのは当然である。この部分が欠落している要求仕様書は、とても合格点を貰うことはできない。しかし「取消」や「追加」などの訂正処理の詳細は、ユーザは書かなくても良いことにしたいというのが、ここの趣旨である。

具体的には、要求仕様書に書かれた「新規」の処理をベースに、「取消」と「追加」の処

理は SE が明確に定義してほしい。もちろんユーザはその検証を、責任を持って実施するというのが前提である。

人間は、間違いを犯す。トランザクションを入力するときも同様である。入力しなければならないトランザクションを入力しなかった、入力しなくて良いトランザクションを入力してしまった、あるいはトランザクションを入力するまでは良かったが、そのトランザクションの内容が間違っていた、などは、現実の事務処理の現場で頻発する間違いである。情報システムには、この人間の間違いを正すために、「訂正処理」の機能が用意されていなければならない。一般にこの訂正処理の機能は、通常の処理の場合より難しいことが多い。

仮に今まちがったトランザクションが入力されたとして、それに対して「取消」のトランザクションが入力されると、結果としてその当初のまちがったトランザクションが入力されなかったのと同じ状態にすることが原則である。最近はオンラインでトランザクションを処理することが多いため、間違って入力されたトランザクションによって処理が進んでしまい、結果として全てを元に戻すことができないケースがある。しかしその場合でも「取消」が入力された後は、元のトランザクションが最初から入力されなかったのと同じ状態であることが、基本的に必要である。

ただし、取消データ入力のタイミングによっては単に元のトランザクションを抹消するだけでは済まず、将来の監査のために訂正の経緯を残したり、経理で反対仕訳を起票して勘定の残高を戻したりしなければならないことがあり、注意が必要である。

「追加」は、正しいトランザクションを後で入力することを意味する。この場合もこの「追加」のトランザクションによって、正しいトランザクションが正しいタイミングで入力されたと同じ状態を作り出すことが原則である。

「訂正」は、入力されたトランザクションの一部の間違いを修正するものである。この処理は非常に複雑になることがあるため、この機能を用意することは止めて、「取消」と「追加」をペアで使用することで、結果として「訂正」を実現することがある。訂正そのものの件数はさほど多くないことが見込めるため、場合によればこの方法での対応で十分なことが多い。

これらの処理の内容をユーザに記述させることは煩雑であるだけでなく、間違いを生み出すことにもなりかねない。ユーザは通常の処理についての要求を明確に、丁寧に記述する。SE はそれを基に、情報システムが果たすべき機能を仕様の形で明確にする。それについて充分な確認を実施した後で、「取消」と「追加」（さらに場合によれば「訂正」）についての仕様を明確にすることが望ましい。

以上が、この要求の背景にある考え方である。ただし遅くともプログラム作成の段階までにユーザは、「取消」と「追加」の仕様、場合によれば「訂正」の仕様について、いずれも正しく定義され、実装されているかの確認を充分に行わなければならないことは、言うまでもない。

IT 関連の専門の設計用語について

アプリケーションについての業界用語が SE に難しいのと同様、IT 関連の専門用語はユーザにはなじみがなく、難しいものである。「SE は業界用語を修得して、ユーザとの間で知識の共有を図るべき」との主張とはいささか矛盾するが、ユーザは IT 関連の専門用語抜きで要求仕様書を記述したい。

この主張の背景には、入出力のプロトタイプなどを提示して貰うことにより、その時点でユーザの要求を明確に SE に伝えることができるという事実がある。要求定義書の記述時点からは随分後ろにずれているが、これらの要求の明確化はこのタイミングでなされても、時間的に充分なはずである。

知識共有のために必要なこと（その２）の詳細分析

ユーザと SE の間で知識共有を実現するために必要な事項については、既に記した。繰り返しになるがその必要な事項は、要求仕様書の「1.3. 専門用語、頭文字語、略語の定義」の部分に適切な記述を行うことである。

この件についてこれまでに記述したのは「結論」だけであるので、以下で何故その結論に到達したのかについて、述べてみたい。

ここで、あるテーマについて双方が知っている／知っていないという軸と、そのことを双方が認識している／認識していないという軸の二軸で整理すると、図表 5-1 のようになる。

図表 5-1　知識共有の状態

	相手の状態を正しく認識している。	相手の状態を間違って認識している、または相手の状態を知らない。
両方が同じ内容を正しく理解している。	◎	b
両方とも理解していると思っているが、内容は一致していない。	a	c
片方は正しく理解しているが、片方は知らない。	a	d
両方とも知らない。	----	----

図表 5-1 で ◎ を付けたケース、つまり「両方が同じ内容を正しく理解している」場合で、さらに「相手の状態を正しく認識している」の場合は、もはや何もする必要がない。これが望ましい状態であり、最終的に全てのケースでこの状態を目指すべきである。

また「両方とも知らない」というケースは、議論する必要がない。この場合そのことが要求仕様書に書かれることがないからである。

したがってこれ以外の状態のものを、どうすれば ◎ を付けたケースに持って行くことができるのかを考えてみたい。

「相手の状態を正しく認識している」、つまりａのケースでは、これまでも述べてきたように、対応は比較的容易である。お互いにじっくりと話をすることで、解決することが期待できる。

しかし相手の状態が不明の場合、つまりｂ、ｃ、ｄの場合は、それほど簡単ではない。ｂの場合は、結果的に問題は起きないのかも知れない。しかし、ｃとｄの場合は簡単ではない。

ここで、ａからｄまでの全てのケースについて何らかの方法で双方の理解／認識を確認する方法として、以下の８種類がある。

① 相手の言動を観察することで、相手の理解／認識を推測する（観察）
② 聞き取り調査をして、相手が理解／認識している内容を得る（聞き取り）
③ 理解／認識内容を網羅的に記述する用紙を用意し、それに記述して貰うことで相手が理解／認識している内容を得る（網羅）
④ 相手の言動から、その根底にある理解／認識を類推する（類推）
⑤ ブレイン・ストーミングなどの機会を用意して、相手の理解を確認する（発想法）
⑥ 理解／内容などを表したモデルを作成し、それを通して理解／認識している内容を得る（表出による確認）
⑦ 相手の言動に表れる矛盾を通して、その根底にある理解／認識の非一致を発見する（矛盾の発見）
⑧ 実際に何かを作り、それを確認することで、相手が理解／認識している内容を確認する（作ってみる／試してみる）

これらの中のあるものはお互いに積極的であり、あるものは消極的である。あるものは相手の出方を単に待っているようなものもある。これらの中で、時間や費用、手間を考えなければ最も望ましい方法は、「作ってみる／試してみる」であろう。しかし前述の通り、実際問題としてこの実現はたいへん困難である。

それに次ぐ好ましい方法は、「表出による確認」である。これはあるモデルを作って、それを通してその理解／認識している内容を表現して貰い、不一致を発見する方法である。

ここで不一致を起こしていると思われるものには、次の二種類のものがあり得る。

- データ、及びそれを構成している項目
- データ（またはその項目）に対して行うべき処理、及びそれに関わる内容

データを明確にするモデルとして、概念データモデルがある。前述の通り概念データモデルは実体関連図（ER 図）、またはクラス図で記述されることが多い。またそれらを記述することは、一般にさほど難しい作業を要しない。

また概念データモデルから新たに抽出された言葉を要求仕様書の「1.3. 専門用語、頭文字語、略語の定義」に追記することによって、言葉のレベルでの理解／認識の一致が一層

進むものと期待できる。

以上が、前述の結論に至った経緯である。

リポジトリについて

要求仕様書の「1.3. 専門用語、頭文字語、略語の定義」に記載した内容について、用語辞書としての「リポジトリ」を作成することを我々は検討した。

リポジトリとは、ソフトウェアを開発する場合の必要な情報や成果物を格納しているデータベースを指す。ここでのリポジトリは、その一部としての用語辞書を我々は想定していた。しかしそれは、ここではその考えを採用しないこととした。

データ項目には、外部から入力されるものとコンピュータ内部で生成されるものがある。リポジトリの中にはこれらのデータ項目について、日本語とプログラムで使用する名称、属性、桁数などの情報が含まれている。これに加えて、コンピュータ内部で生成されるデータ項目については、生成（あるいは計算）の方法も併せて格納したい。そしてこの情報を使用して、データ加工のプログラムが自動的に生成されるところまで最終的には持って行きたい。

しかし現状は、そのプログラムの自動生成まで実現することは困難と判断する。そして、単なる用語辞書としての使用にとどまるのなら手間をかけてリポジトリを電子化するには及ばない。これが、我々が今リポジトリの作成を見送った理由である。

近い将来には我々はリポジトリを作成し、加工プログラムの自動生成まで実現したいものである。

（注）

1．このプロジェクトの提案書や企画書では、このテーマには「暗黙知の整理」という標題が使われている。しかし「暗黙知」という言葉は野中郁次郎教授らの定義（[NON96]、[NON03]）が広く行き渡っており、我々の検討のこのテーマについてはこの言葉を使わない方がよいとの議論が我々の検討の過程であった。この議論の結果を受けて、この章の標題を標記の通りに変更した。

2．この章の標題は、データベース理論の立場からすれば「意味共有の方法」とするのが正しいのかも知れない。しかしここで使われている「意味」という言葉はたいへん難しいので、もっと一般的な「知識共有の方法」という言葉に変えた。

参考文献

[BRO75] フレデリック・P・ブルックス，JR 著、滝沢徹他訳、「人月の親和 狼人間を打つ銀の弾はない 原書発行 20 周年記念増訂版」、アジソン・ウェスエイ・パブリッシャーズ・ジャパン（株）、1996 年.

[IEE98] IEEE-SA Standards Board, “IEEE Recommended Practice for Software

Requirements Specifications IEEE Std 830-1998”, The Institute of Electrical and Electronics Engineers, Inc., 1998.

[NON96] 野中郁次郎、竹内弘高著、「知識創造企業」、東洋経済新報社、1996 年.

[NON03] 野中郁次郎、紺野登著、「知識創造の方法論 ナレッジワーカーの作法」、東洋経済新報社、2003 年.

６．要求仕様の定義方法

この章の目的

既に第 3 章（「3. 要求仕様書とは」）で述べたように、我々は清水氏が提唱する「USDM 表記法」を用いて要求仕様書を記述することとした。我々は基本的に、この立場を保持するつもりである。しかし仮に USDM 表記法が完全なものでないなら、第 3 章で定めた我々の要求仕様書に何らかの手を加える必要があるかも知れない。そしてこの世の中には、残念ながら完全なものは存在しない。

この章ではこの立場から、先ず要求仕様書が果たすべき役割を考え、それを USDM 表記法に当てはめた場合に見つかる不満足な点を明らかにし、その上でそれらを解決するために、我々の USDM 表記法に何を付け加えなければならないのかについて議論をしたい。

要求仕様書の使い方

情報システム開発における要求仕様書の位置づけは、既に第 3 章で述べた。つまり要求仕様書はその情報システムについて最初に作成される成果物であり、その情報システム構築に必要な全ての作業がこの成果物を出発点にしている。具体的に我々は、要求仕様書を以下のように使うことになる[POW01]。

- 議論のたたき台として
- コミュニケーションの手段として
- 作業のための文書として
- アーキテクチャ設計の出発点として
- プロトタイプ作成の出発点として
- サイズ見積もりの入力として
- コスト見積もりの入力として
- 設計の入力として
- プロジェクト計画立案の入力として
- テストの入力として
- ユーザマニュアル作成の入力として
- 解決探しのヒントとして
- ユーザが何を手に入れるかを示すため
- 提案を評価するため
- 合意を文書化するため
- 合意を得るため
- 知識を整理するため
- 理解を表現するため

使い方についての具体的な１つ１つの議論よりも、ここではその使い方の多様性に注目し

ておきたい。

作成時に要求仕様書に求められる品質

このような多彩な使い方がされる要求仕様書であるから、それに求められる品質は必然的にたいへんシビアなものとなる。これを、作成時、検証時、更新時、及び参照時に分けて、それぞれ考えてみたい。

先ず作成時についてであるが、次のような要求を挙げることができる。

- 粒度の統一を図ることができること
- 客観的な記述方法を実現できること
- 先行文書との関係を表示できること
- 漏れ／抜け、矛盾、ダブりを発見しやすいこと

粒度の統一を実現する原則として、「1 つの文章には動詞を 1 つ」というものがある。この粒度の統一と客観的な記述は、簡単とはいわないにしても、現在の USDM 表記法で対応可能である。つまりこれらの要求へは、作成者が作成時にこれらの点に留意して記述するしか対応の方法がない。

しかし先行文書との関係の表示は、現在の USDM 表記法には盛り込まれていない。これはトレーサビリティの問題の 1 つと捉え、要求仕様書を出発点にして設計やプログラムへのトレーサビリティを考える際に、一緒に議論したい。

漏れ／抜け、矛盾、ダブりの発見について、清水氏は特別の対策を提示していない[SIM05]。仕様が明確に整理されて記述されているなら、SE はその仕様を容易に把握する技術を持っている。その結果として漏れ／抜け、矛盾、ダブりは、特別の方策を考えなくても発見できるというのが彼の立場である。確かに、その通りかも知れない。しかし仮にその通りであるにしても、我々はこれらを発見する方策を別途持つことが必要であろう。このような立場から、この問題は後で議論したい。

検証時に要求仕様書に求められる品質

ここでの検証とは、レビューを意味している。そのレビュー時に求められる品質として、次のような要求を挙げることができる。

- 任意の事柄に関係する要求仕様を一覧にできること
- 要求の裏側にある「理由」を記述できること
- 検証済みかどうかの状態の管理が可能であること

簡単に対応できるかどうかはともかく、我々の USDM 表記法は Excel を使って記述することにした。このことから最初の要求（任意の事柄に関係する要求仕様を一覧にできること）は、今のままでも対応できるものと考える。

２つ目の「理由」の記述は、これこそまさに我々が USDM 表記法を選んだ理由の最大のものの１つであり、今の記述法で充分に対処可能である。

3 つ目の「状態の管理」については、Excel では必要に応じて列を追加し、その状態管理のために使うことができる。したがって、これも既に解決済みと考えたい。

更新時に要求仕様書に求められる品質

更新とは要求仕様書に、変更などのために手を入れることを指す。

要求仕様書は情報システムが開発されている間だけ存在する文書であって、保守などのためにこの文書がそのまま使われることはないと既に述べた。しかしそれでも要求仕様書は、変更されなければならないとも述べた。

このような文書の変更に対応するためには、ソフトウェアの構成管理の仕組みを導入することが望ましい[ISO98a]。それはともかくとして、更新に備えて要求仕様書に求められる品質には次のようなものがある。

- 変更範囲を特定しやすいこと
- 文書を修正しやすいこと
- 版管理が可能であること

最初の 2 つの要求も、作成者が作成時に注意するべき事項に属する。しかし我々は何度も述べるとおり、要求仕様書を Word ではなく Excel を使って作成しようとしている。ワードプロセッサで文書を作成する場合、ともすれば文章は散漫になりかねない。しかし Excel では感覚的に、「箇条書き」に近い形で文章を作成することができる。このためはじめの 2 つの要求には、比較的対応しやすいと考える。

版管理の問題は、ソフトウェアの構成管理のテーマとして「８．要求仕様を設計からプログラムまでトレースする方法」で議論することとしたい。

参照時に要求仕様書に求められる品質

この章の最初にも述べたように、要求仕様書は参照されることが多い。このため、要求仕様書には参照時に求められる品質がことのほか重要である。

参照時のための品質として、次のようなものを挙げることができる。

- 合意を明確にするための文書として、記述が曖昧でないこと
- 見積もりや計画への入力として、機能に優先順位が付けられていること
- アーキテクチャを含む設計への入力のために
  - スコープの記述ができること
  - 品質目標が明示できること
  - 設計書やソースプログラムなど他の文書との関係を示すことができること
  - 任意の事柄に関係する要求仕様を一覧にできること
- 要求仕様書の記述が詳細になりすぎないこと

ここであげたものの中には既に解決済みのものがあり、あるいは作成者が作成時に考慮するしか方法がないものもある。しかし機能の優先順位付けの問題は、まだ提起されていな

かった。したがってこの問題を、後で議論することにしたい。

また要求仕様書の記述が詳細になりすぎないようにする件は、記述時の粒度の問題とも関わりを持つ。この回避策も、後で検討したい。

仕様間の関係

アラン・デーヴィスは、仕様間に依存関係がある場合があり、それらを要求仕様書に記述しておくことで機能の追加／削除への対応が容易になると指摘している[DAV05]。さらに清水氏から、これによってさらに将来の仕様変更への対応に威力を発揮するとのコメントを頂いた。

したがってこの仕様間の依存関係について、後で議論することにしたい。

USDM 表記法に追加するべき項目

これまでの要求仕様書に求められる品質などについての議論から、第3章で述べたUSDM表記法に、我々はさらに次の項目を追加する必要があるということになる。

1. 漏れ／抜け、矛盾、ダブりの発見法を用意すること
2. 機能に優先順位を付ける方法を用意すること
3. 記述が詳細になりすぎることを避ける方策を用意すること
4. 仕様間の依存関係を考慮すること

この章のこの後の部分で、この４つの問題を議論したい。

漏れ／抜け、矛盾、ダブりの発見法

要求の漏れ／抜け、矛盾、ダブりの発見法と標題には一律に記述したが、漏れと抜けの発見は、非常に難しい。これは要求仕様書のレビューを通して、その関係者が対応するしか方法がない。清水氏は前述の通り、要求の階層化などによってゴールとしての範囲が適切に狭められた状況の下で要求が適切に記述されていると、SE は漏れや抜けを見つけることができる能力を持っていると言っている。ここでは、それに大いに期待をしたい。したがってこの後では、矛盾とダブりの発見法について議論することとする。

要求仕様書の作成に関わる問題の解決に、品質機能展開（QFD： Quality Function Deployment）の手法を適用しようとする考え方がある[RYU06]。その QFD の手法の 1 つに、この矛盾とダブりの発見法がある。

具体的には図表 6-1 に示す表を作成し、行と列を相互に参照しながら矛盾とダブりを発見する方法である[JIS03]。

例えば行と列で示された機能間に矛盾がある場合には対応するセルに × 印を、ダブりがある場合は △ 印を記述する、などの方法で矛盾とダブりを発見し、表示することができる。

しかし仮に機能の数が 100 あるとすると、約 5,000 件のチェックが必要になる。作業量

と得られる成果とのかねあいで、この方法で作業するかどうかを検討する必要がある。

この例に近いサンプルとして、武蔵工業大学の横山先生から頂いた教育支援システムの機能をまとめたものがあるので、付 6-1 としてこの章末に添付する。

機能1										
機能 2										
機能3										
・										
・										
・										
・										
機能 n−2										
機能 n−1										
機能 n										
	機能1	機能2	機能3	・	・	・	・	機能n−2	機能n−1	機能n

図表 6-1　機能の矛盾／ダブりの発見法（[JIS03]より）

なお図表 6-1 と付 6-1 はいずれも、行と列に同じ機能を列挙する場合である。しかし行に記述するものと列に記述するものを変えて QFD のこの表を作ることで、別の用途に用いることができる。例えば行にワークタスクを、列に情報システムの機能を記述してこの表を作成すると、どの作業（ワークタスク）で情報システムのどの機能を使うかを明示することができる。

機能に優先順位を付ける方法

第 3 章で述べた USDM 表記法には、要求に全く差がついていなかった。別の言い方をすると、要求は全てフラットだった。

しかし現実の要求が全てフラットということはないし、その要求を情報システムの上に具現化しようとするとき、ある要求は別の要求より優先順位が高く、ぜひとも実現して欲しい、というようなものがある。

したがってここで USDM のテンプレートを一部変更し、要求の優先度を表示する列を設けることとする。

優先度そのものは、あくまで業務上の優先度とし、次に示す三段階で表現することとする。

優先度：1　必須機能

優先度：２　重要な機能

優先度：３　あると良い機能

ここで「必須機能」とは、それが含まれなければシステムの存在意義を認めることができない必要最低限の機能で、以下のようなものを指す[JUA07a]。

- 多くのタスクを支援し、さらにビジネスゴールに直結する機能
- ユーザのニーズに根ざした確かな根拠がある機能
- 当該システムを意義あるものとして使用するための中心的な機能
- プロトタイプを作成し、ユーザビリティ・テストを開始するために最低限必要な機能

また「重要な機能」とは、システムをリリースするために最低限必要な機能である。しかしこの機能を明確に、一般論で表現することは難しい場合が多い。

「あると良い機能」とは、ユーザの満足度や使いやすさには貢献する。しかしさしあたって存在しなくても、システムとして成立しうる機能である。システムを使いやすくするための様々な機能のアイディア群が、これに含まれる。ここから必要に応じて、「重要な機能」に格上げされるものが出てくる。

このように書くと、あるいは優先順位付けはさほど難しくなく見えるかも知れない。しかし時として、ステークホールダ間で優先順位に差があったり、その結果それが衝突したりすることがあり得る。このような場合、これを調整する何らかの方法が必要になる。この調整のための委員会を持つというのは１つの方法であるし、「システム要求」の結論に戻るというのも別の方法と考えられる。

いうまでもなくこの優先度は、開発段階の優先順位付けであり、フェーズ分けや開発規模が増大した場合にカットの対象にする機能を洗い出すためにも用いられる。しかしここで最優先の優先順位を付けた機能が、常にフェーズ 1 で稼働することを意味しているわけではない。開発の手順によって、データベースの更新手順などここでは見えなかった機能をもっと優先して開発しなければならないことがあり得る。

記述の詳細化を避ける方策

このテーマは「要求仕様書として何を書くべきか」、「要求仕様書にどこまで書くべきか」という、たいへん本質的な議論と関わりを持つ。また記載に当たっての、粒度の統一とも関わりを持つ。

建前を言えば、この問に対する答えは、「記述は可能な限り詳細に、漏れなく記述するべし」ということになる。しかしその一方で、あまり詳細な記述が延々と続く要求仕様書は読むのに苦痛である。詳細さの陰に隠れて、重要な要件を見逃してしまう危険性もある。これは、何としても避けたい。

ここで、要求仕様書を書く立場で記述を詳細にしたくなる対象は、入出力とそれに関連したデータ加工に関わる内容と推察する。したがって我々は要求仕様書とは別に、データ加

工まで記述できる入出力の定義書をこの段階で併せて作成することを提案する。

このような文書は、既に各社のソフトウェア開発に関わる標準化の体系の中に位置づけされていると考える。しかしそれを記述するのは、設計作業の中での外部設計の段階となっていると思われる。入出力のイメージなどを完成させての入出力の定義書の完成は、外部設計段階でよい。しかし入出力されるデータ項目やデータ加工の方法などを記述することは、遅くともこの要求仕様書作成段階から始めることとしたい。

「４．要求仕様書作成に関わる作業の流れ」での議論の中で、この資料の作成はシステム企画（ビジネス検討）から始めることが望ましいと述べた。そして入出力定義書に記載する内容をこの要求仕様書に重複して記載はしないと決めることで、要求仕様書の記載を高いレベルで統一することができると我々は考える。

入出力定義書に記載するべき項目の例を「付４．画面／帳票定義書の記載項目」としてまとめ、この報告書の巻末に添付する。ちなみに、入出力定義書も更に詳細にしない工夫として、組織で入出力の方法についての標準の方法を決め、基本は全ての入出力でそれを守ることとし、特別にその基本と異なる入出力を行うときだけ、入出力定義書に記述をするという方法を提案したい。

仕様間の依存関係

アラン・デーヴィスはその著「成功する要求仕様失敗する要求仕様[DAV05]」の中で、仕様同士は常にいつも独立ではなく、場合によれば双方の間に依存関係があると記している。そしてこの依存関係には、次の 5 つの種類があると彼は言う。

1. 必須依存
2. 作業依存
3. サブセット依存
4. 網羅依存
5. 価値依存

最初の必須依存とは、1 つの仕様を満たして、初めてもう 1 つの仕様を満たす意味が出てくるような関係を言う。具体的には、次のようなケースを挙げることができる。

仕様 A：システムは、顧客に機能 X を提供する。

仕様 B:システムは、顧客が機能 X を使用している間、1 分間に 100 円ずつ課金する。

2 つ目の作業依存とは、仕様 B を実装すれば仕様 A の実装が簡単になる関係を言う。具体的には、次のようなケースを挙げることができる。

仕様 A:システムは、売上げ計上日から 60 日以上経った売掛債権の帳票を作成する。

仕様 B：システムは、一般的な帳票生成ユーティリティを提供する。

3 つ目のサブセット依存と 4 つ目の網羅依存は、USDM では要求と仕様の関係で対応されているので、説明を省略する。

最後の価値依存とは、ある仕様の要求を満たすことによって、別の仕様へのニーズが強ま

る、あるいは弱まることをいう。

アラン・デーヴィスは、この 5 種類の仕様間の依存関係の明細を記述できるように、要求仕様書にそれぞれの欄（列）を設けることを提案している[DAV05]。

しかし我々は検討の結果、この提案を受けないこととした。仕様間に依存関係があることは理解できる。しかしその依存関係を彼が提案するような形で記載することが可能なのかということの判断ができなかったというのが、その理由である。

仕様間の依存関係を記載する必要がある場合、USDM 表記法に取り込むのではなく別の資料を用意する、あるいは USDM 表記法の備考欄を使用する、といった方法で対応することとしたい。

ユースケース図を併用するか

オブジェクト指向技法に則って情報システムを分析／設計する場合に用いられるモデル類をまとめた UML（Unified Modeling Language）に、ユースケース図がある[BOO99]。ユースケース図以外のモデルは基本的に、情報システムを構築する立場、つまり SE が使用するモデルであるが、ユースケース図は情報システムを使用する立場から書かれたモデルで、他のモデルと比べるとかなり異色であると評価できる。これを USDM 表記法と併用するのがよいのだろうか。

これへの解答は、基本的に要求定義書を作る SE の考え方の問題に帰着する。しかし仮に記述するとしても、あまり詳細なレベルでユースケース図を記述する必要はなく、要求の最も上位のものだけを記述し、USDM で記述された全体の機能の「目次」か「索引」のような形で使用することを我々は提案したい。

またユースケース記述をユースケース図と併せて記述するかどうかも、SE の考え方の問題である。第 12 章で示した我々の例題には、どちらも USDM 表記法の索引レベルで記述してみた。

さらに情報システムが持つべき機能を階層化して表示する機能階層図という図面も、必要に応じて使用するのがよい。

第 12 章に示した我々の例題では、画面の流れも記述してある。しかしこれも、ここの段階ではオプションとしておきたい。最終的にはこの図は基本設計の段階で完成されるべきものであって、この段階で作成することは少し早いかも知れない。

我々の要求仕様書

第 3 章に示したテンプレートに、この章で述べたことを追加／修正を施した我々の最終的なテンプレートを、この巻末に「付 3. 要求仕様書のテンプレート」として表示する。

また我々は、このフェーズで途中まで作成した「画面／帳票定義書の記載項目」を我々の要求仕様書の一部として位置づける。そこに記載するべき項目は、「付 4. 画面／帳票定義書の記載項目」としてまとめた。

参考文献

[BOO99] グラディ・ブーチ著、オージス総研オブジェクト技術ソリューション事業部訳、「UML ユーザーガイド」、(株) ピアソン・エデュケーション、1999 年.

[DAV05] アラン・M・デービス著、萩本順三他監修、高嶋優子訳、「成功する要求仕様失敗する要求仕様」、日経 BP 社、2006 年 11 月 6 日.

[ISO98a] 日本工業標準調査会情報部会審議、「ソフトウェアライフサイクルプロセス－構成管理 TR X 0018：1999 (ISO／IEC TR 15846：1998)」、(財) 日本規格協会、平成 11 年 10 月 30 日.

[JIS03] 日本工業標準調査会審議、「マネジメントシステムのパフォーマンス改善－品質機能展開の指針 JIS Q 9025：2003」、(財) 日本規格協会、平成 15 年 2 月 20 日.

[JUA07a] 「システム・リファレンス・マニュアル 2007 年版」(社) 日本情報システム・ユーザー協会、2007 年. (未刊行)

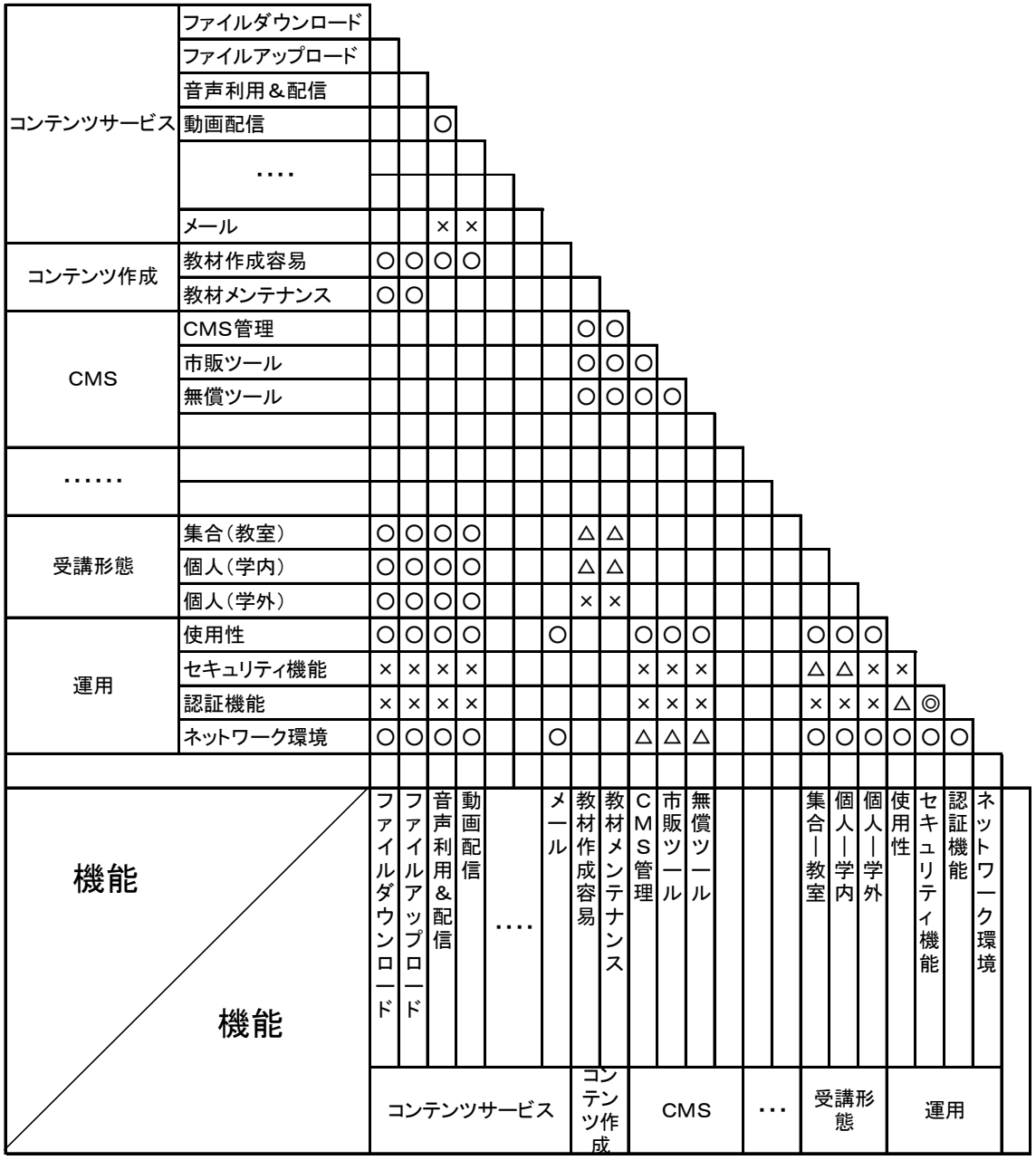
[POW01] Power, N., "Variety and Quality in Requirements Documentation," Proc. of REFSQ2001, 2001.

[RYU06] 劉功義、「プロジェクト計画における要求整理方法の提案 プロジェクトマネジメント学会 2005 年度第 2 回研究員会フォーラムでの講演資料より」、PM 学会研究委員会、平成 18 年 3 月 1 日.

[SIM05] 清水吉男著、「[入門+実践]要求を仕様化する技術表現する技術～仕様が書けていますか」、(株) 技術評論社、平成 17 年.

付 6-1　QFD の適用例

教育支援システム（例）



◎ or ○　機能同士が相乗効果が見込めるもの
× or △　機能同士が競合するもの(背反特性)
CMS:Contents Management System

7．ユーザ部門と IT 部門の役割

ユーザ部門と IT 部門の関係の変化

少し話が大きくなるかも知れないが、ここで企業の情報システムの形態の変化から考えてみたい。情報システム化の歴史の長い企業はこれまで、情報システムの形態について以下のような変化を経験してきた。

- 1980 年代まで：ホスト集中
- 1990 年代：分散機、及びそれをベースにしたクライアント／サーバ方式
- 2000 年以降：パッケージの活用、及び Web コンピューティング

ホスト集中の時代は、IT 部門主体で情報システム化が進んだ時代だった。しかし一般にバックログが大量に滞積し、ユーザ部門の欲求は満たされない状態が続いていた。この時代を、ユーザ部門と IT 部門の関係の第一世代と位置づけたい。

この反動が、1990 年代に入って表面化する。つまりそれまで IT 部門中心で進んでいた情報システム化に対するユーザ部門の声が大きくなり、ダウン・サイジングとオープン・システム化の進展とも歩調を合わせて、極端な場合は IT 部門抜きで、ユーザ部門が直接アウトソーシングを行って情報システムの構築を行ったケースもあった。この時代を、ユーザ部門と IT 部門の関係の第二世代としておこう。

この第二世代の影響はさほど時間をおかずに、情報システムの品質低下や保守対応の困難さという形で表面化する。ここでこの進め方についての見直しが入り、両者が協調して情報システム化を進めることが必要という機運が生じることになる。これをユーザ部門と IT 部門の関係の第三世代と見なせる。今はまさに、この第三世代の入り口から少し奥にまで進んだ状態と定義できる。

第三世代におけるユーザ部門と IT 部門の関係

第三世代におけるユーザ部門と IT 部門の関係を一言で表せば、前述の通り「協調体制」という言葉がふさわしい。

例えば情報システム化実施の決裁を経営者に求める場合、ユーザ部門は情報システム稼働後の効果の実現に責任を持ち、IT 部門は情報システムの構築に当たってスケジュールと予算、及び製品の品質に責任を持つという分担で当たることが普通になっている。

今の我々のテーマである要求仕様書作成についていえば、要求をユーザ部門が記し、それを基に仕様を IT 部門が付け加えるという形になる。しかし要求仕様書作成の作業だけを取ってみても、この 1 行で片付けてしまえるような簡単なものではない。要求を確定するまでに、現行システムの機能を洗い、問題点を抽出し、その解決方法を検討し、というような長いプロセスがあって、その後でやっと新システムへの要求が固まる。

この要求仕様書作成の前の「ビジネス検討（システム企画）」から、開発に決裁がなされて開発計画の具現化からプロジェクト開始に至る一連のプロセスを考えると、ユーザ部門

と IT 部門が共同で、非常に多くの作業をこなさなければならないことになる。

ユーザ部門と IT 部門の役割分担

このような多くの作業を共同で、無駄なく、滞りなくこなしてゆくためには、単に「協調体制」という言葉だけでは不十分である。このために進んだ企業では、現在次の 3 つのことを実施している。

- ユーザ部門と IT 部門の役割分担を、何らかの形で明確にしている。
- ユーザ部門と IT 部門がそれぞれ行うべき作業を規定した規程やマニュアル類を整備し、さらに作業を行うときに参照できるようサンプル類を準備している。
- ユーザ部門はやはり情報システム化の推進で不慣れなところがあるため、IT 部門内にユーザ部門支援の部隊を配備したり、ユーザ部門向けの教育活動を積極的に推進したりしている。

このプロジェクトの会員企業の 1 つが使用している役割分担の表を、この章末に付 7-1 として添付する。また JUAS も同様のモデルを既に発表しているので、それを付 7-2 として添付する[JUA05]。

規程類の整備

やはりこのプロジェクトのメンバーの企業の 1 つは、「指針」と呼ばれるレベルのものを 9 種類、「手引き」と呼ばれるレベルのものを 29 種類、「ガイドブック」と呼ばれるレベルのものを 45 種類、合計で 83 種類の規程類を保持している。「指針」とはハイレベルの方針を記述したもの、「手引き」とは具体的な作業の方法を記述したもの、「ガイドブック」とはその作業を行う場合に参照するものとして、使い分けている。

また別の企業では、ユーザ部門と IT 部門の役割責任を明確にするとともに、それぞれの部署が責任をもって作成しなければならないドキュメント類を定め、要件定義における検討ポイントや記載要領等を解説したガイドを準備し、経験の浅い部署も理解できるように具体的なサンプル集を用意し運用をしている。なおサンプル集については IPA／SEC のホームページのユーザ事例検索システムに掲載されている。

ユーザ向けの教育活動

やはりこのプロジェクトのメンバーの企業の 1 つが実施しているユーザ部門向けの教育活動の概要を、この章末に付 7-3 として添付する。

またこのプロジェクトの他のメンバーの企業では、IT 部門の中に「コンサルタント部隊」と呼ぶ情報システム化の初期段階でユーザ部門を支援する組織があり、効果を挙げている。

やはりユーザ部門は情報システム開発を専門にしているわけではないので、教育や支援を充分に行わなければ、いくら役割分担を明示してもそれだけでは効果が上がる活動を期待することが難しいというのが現状であろう。前向きの、積極的な取り組みと評価できる。

情報システム化を成功させるために

ユーザ部門と IT 部門の関係の第三世代は、第一世代と第二世代で生じた問題をこれから発生させないために作られた体制である。これを既に「協調体制」と名付けたが、協調体制とは即「責任遂行体制」と言い換えても良い。

ユーザ部門はユーザ部門として、IT 部門は IT 部門としての役割と責任があり、第三世代ではそれが明確になっている。その役割と責任をお互いに共同で遂行することが、情報システム化で成功を勝ち取る唯一の道であることを、両部門とも改めて意識していただきたい。

ただし情報システムの化の推進は、ユーザ部門と IT 部門だけの共同作業で進めるものではない。しっかりと経営者を巻き込み、これらの組織に責任を持つ人たちでステアリング・コミティーを作って、ここで大所高所からのチェック機能を働かせ、必要なら的確に方向付けを行うことが必要である。例えば、各フェーズの作業結果とこの時の状況をレビューして次のフェーズに移って良いかどうかの判断を行うなどというのが、このコミティーの機能の 1 つとなる場合がある。

あるいは、1 つのフェーズは 1 つ以上の PDCA サイクルで形成される。この P と D は付 7−1 や 7−2 の示したユーザ部門と IT 部門で行うとして、C と A はこのコミティーが、あるいはこのコミティーが任命する組織が実施することが必要かも知れない。

ユーザ部門の管理者の役割

第 2 章で、「ユーザの担当者が変わると要求（仕様）が変わる」とか、「要求仕様の提示が遅れる」といった問題があると指摘した。またこれ以外に、要求仕様書に書かれた要求を設計段階などでより詳細にする必要が生じることがあり、この詳細な要求が組織確認されておらず、ユーザ部門の担当者独自の意見であるという場合がある。

また、ユーザ部門の管理者はこの情報システム化を契機にある業務改善を進めたいと念願していても、その思いが作業の現場に的確には伝わっておらず、作業現場の人がテストに参画して、新しい情報システムの機能やそれ以前の開発のスタンスに拒否反応を示すようなことがある。

現実に、これらの問題の解決は非常に難しいものであろう。しかしこれらの問題を起こさないようにすることはユーザ部門の管理者の役割であることを、ここで明確にしておきたい。ユーザ部門の管理者がこの役割を明確に果たすために、IT 部門も必要な協力を行わなければならない。

注）

この章は、当初は「利用部門、IT 部門の責任者によるシステムコンセプトの確認内容の明確化」という標題を予定していた。しかし議論の過程で、標題を「ユーザ部門と IT 部門

の役割」と変更した。

参考文献

[JUA05] 「システム・リファレンス・マニュアル（SRM)」、(社) 日本情報システム・ユーザー協会、2005 年 9 月.

付 7-1　ユーザ部門と IT 部門の関係１（メンバー企業の事例）

工程	ステップ（上段） 目的（下段）	作業内容	ドキュメント名	ドキュメント サンプル	○：検討の主体	
					業務部門	システム部門
企画	A-1 変革目標の設定 経営からのニーズ、当社を取り巻く環境の変化、業務実態等から、潜在する課題をも的確につかみ、業務改革の必要性を認識し、変革目標を明確化する	A-1-1 変革要因の整理	調査表（内部環境）	CA-1	○	
			調査表（外部環境）	CA-2	○	
		A-1-2 現状課題分析	調査表（利用者ニーズ）	CA-3	○	
			調査表（システム利用実態）	CA-4		○
			現行業務フロー（パターン）	CA-5	○	
			現行業務フロー	CA-6	○	
			現行業務ルール	CA-7	○	
			現行業務組織関連図	CA-8	○	
			現行システム概要	CA-9		○
			現行システム機能一覧	CA-10		○
			課題一覧表	CA-11	○	
		A-1-3 変革目標の設定	戦略マッピング	CA-12	○	
			最終的な目標と評価指標	CA-13	○	
			変革目標一覧表	CA-14	○	
			プロジェクト撤退・縮小ルール	CA-15	○	
	A-2 変革方針の具体化 業務確信を実施する対象業務、対象箇所を明確に定義するとともに、ITの活用を含めた新しい強い業務イメージを検討する	A-2-1 新業務イメージの立案	新しい業務イメージ	CA-16	○	
			新業務組織関連図	CA-17	○	
			システム利用イメージ図	CA-18		○
			発明考案届出書	CA-19	○	
		A-2-2 変革対象範囲設定	変革対象範囲一覧	CA-20	○	
		A-2-3 制約事項の確認	機密保護対策表	CA-21		○
			プロジェクトリスク一覧表	CA-22	○	
			制約事項一覧表	CA-23	○	
		A-2-4 変革アプローチの決定	変革基本方針	CA-24	○	
	A-3 投資計画の立案 業務革新の効果を予測するとともに、基本検討以降を実施するにあたって検討体制および検討スケジュールを作成する	A-3-1 投資効果の想定	効果内訳表	CA-25	○	
			概算費用内訳表	CA-26		○
		A-3-2 課題およびリスク要素の抽出	プロジェクト撤退・縮小ルール	CA-15	○	
			プロジェクトリスク一覧表	CA-22	○	
		A-3-3 投資計画立案	実施体制	CA-27		○
			役割分担表	CA-28		○
			実施スケジュール	CA-29		○
基本検討	B-1 ソリューション検討 変革目標を個別業務単位での改善目標に細分化するとともに、新しい業務イメージを業務プロセスレベルまで詳細化しつつシステム化すべき範囲とその優先順位を明確にする	B-1-1 現行業務棚卸	調査表（内部環境）	CB-1	○	
			調査表（外部環境）	CB-2	○	
			現行業務ルール	CB-3	○	
			現行業務棚卸表	CB-4	○	
			現行業務フロー（パターン）	CB-5	○	
			現行業務フロー	CB-6	○	
			課題一覧表	CB-7	○	
			現行インプット一覧	CB-8		○
			現行アウトプット一覧	CB-9		○
			現行DB一覧	CB-10		○
			現行データ量	CB-11		○
		B-1-2 新業務プロセス検討	新業務フロー（パターン）	CB-12	○	
			新業務フロー	CB-13	○	
			新業務フロー（詳細）	CB-14	○	
			新業務ルール	CB-15	○	
			新業務組織関連図	CB-16	○	
			主要機能IPO図	CB-17		○
			変革目標一覧表	CB-18	○	
			変革対象範囲一覧	CB-19	○	
			戦略マッピング	CB-20	○	
			最終的な目標と評価指標	CB-21	○	
		B-1-3 システム化主要項目検討	システム利用イメージ基本検討図	CB-22		○
			システム連関図	CB-23		○
			システム化機能一覧表	CB-24		○
			主要インプット・アウトプットイメージ	CB-25		○
			新インプット一覧	CB-26		○
			新アウトプット一覧	CB-27		○
			新DB一覧	CB-28		○
	B-2 開発要件・優先順位検討 必要なシステム化機能に関して、法制度等の課題を整理し、システム技術、性能目標等の適用の方向性を明らかにする	B-2-1 開発要件整理	性能目標設定	CB-29		○
			ハードウェア・ネットワーク構成基本検討図	CB-30		○
			ソフトウェア構成基本検討図	CB-31		○
		B-2-2 システム課題の整理	適用技術一覧	CB-32		○
			機密保護対策表	CB-33		○
			障害・災害対策表	CB-34		○
			運用要件一覧表	CB-35		○
			移行要件整理表	CB-36		○
			業務管理リスト	CB-37		○
		B-2-3 開発優先順位検討	開発方針	CB-38		○
	B-3 開発計画の策定 業務革新の効果の確認をするとともに、投資上限額を設定し基本設計以降の開発計画を示す	B-3-1 投資対効果精査	効果内訳表	CB-39	○	
			開発規模見積もり	CB-40		○
			ハード等設備費	CB-41		○
			ソフト等購入費	CB-42		○
			ランニング費用	CB-43		○
			概算費用内訳表	CB-44		○
		B-3-2 リスク整理と対策	プロジェクト撤退・縮小ルール	CB-45	○	
			プロジェクトリスク一覧表	CB-46	○	
		B-3-3 開発計画の作成	実施体制	CB-47		○
			役割分担表	CB-48		○
			稼動後の体制図	CB-49		○
			実施スケジュール	CB-50		○

この表は、独立行政法人情報処理推進機構（IPA）ソフトウェアエンジニアリングセンター（SEC）の「超上流から攻める IT 化の事例検索システム」から引用した（https://sec.ipa.go.jp/enterprise/index.php）。

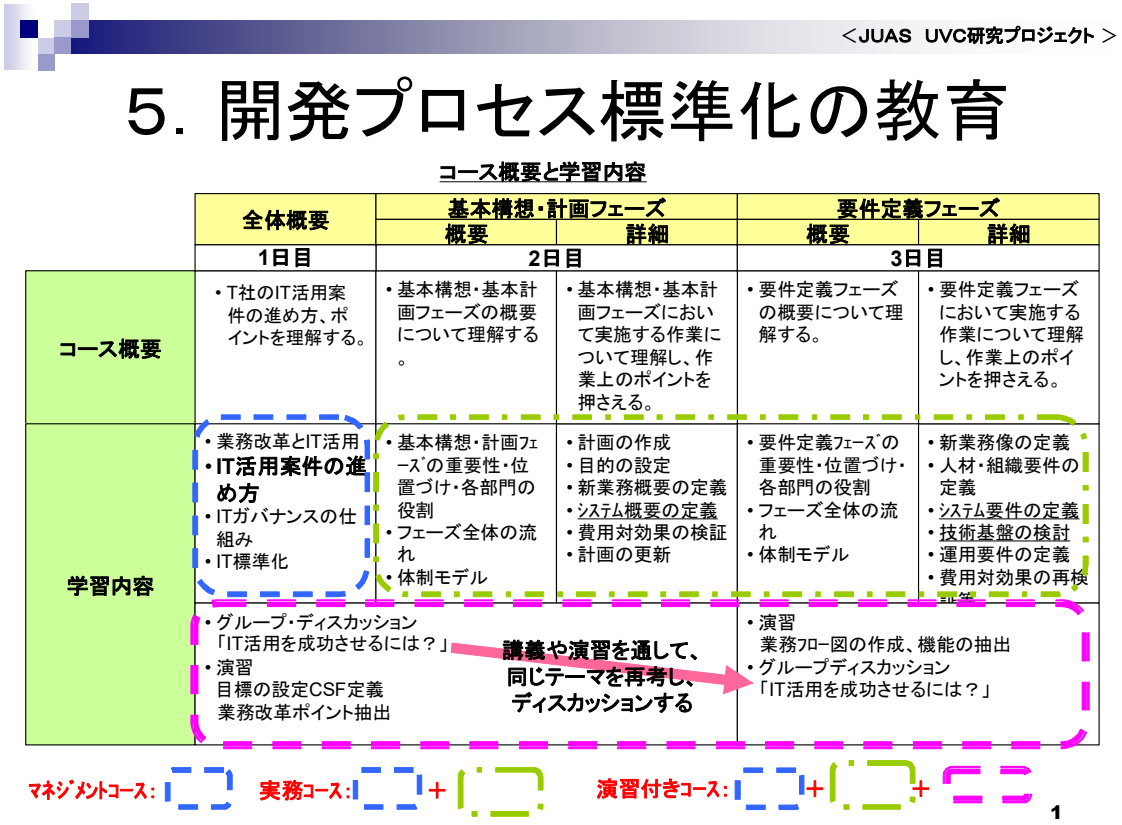
付 7-2　ユーザ部門と IT 部門の関係２（JUAS のモデル）

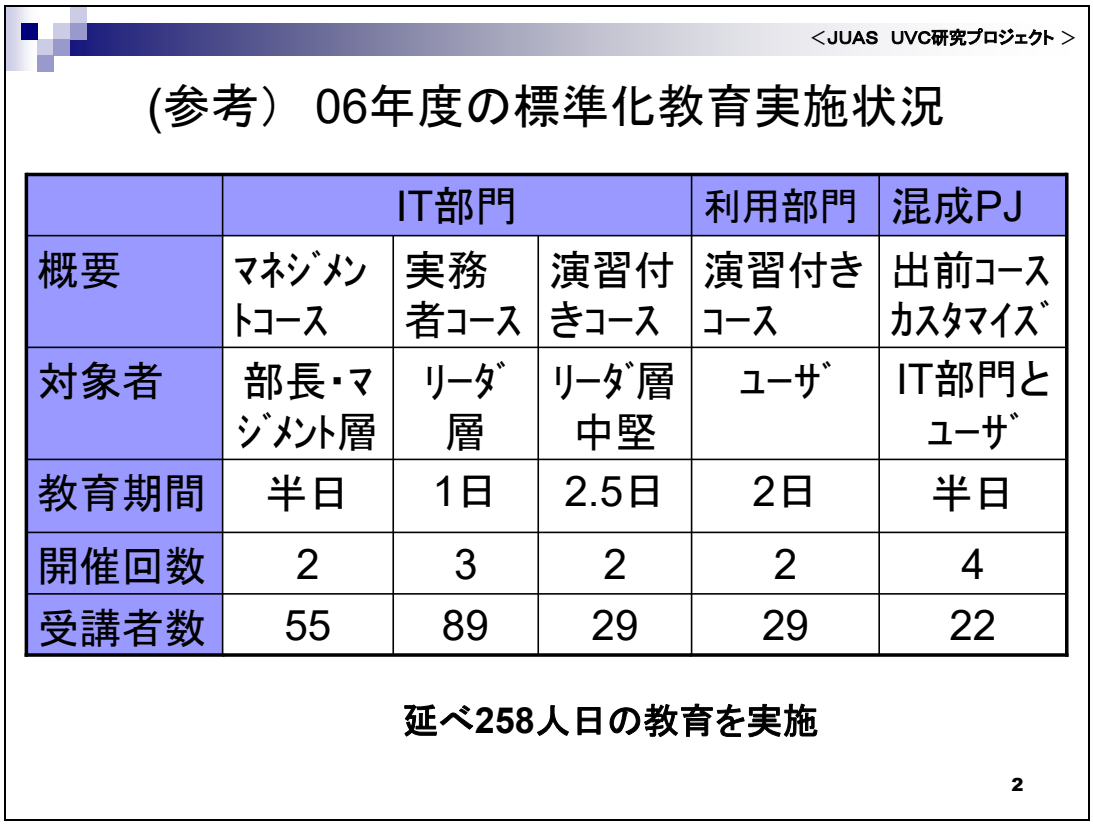
基本項目	内容	実行責任部門	
		利用部門	IT 部門
①システムの概要	システム化の目標・方針	◎	○
	狙いとする効果	◎	○
	運用対象者	◎	○
	既存システムとの関係	○	◎
②提案依頼手続き	説明会日程		◎
	対応窓口		◎
	提供する資料	○	◎
	参加資格条件		◎
	提案手続き		◎
	選定方法		◎
③依頼事項	システム化の依頼範囲（業務関係）	◎	○
	（システム関係）	○	◎
	依頼内容、業務の詳細（業務関係）	◎	○
	（システム関係）	○	◎
	システム構成	○	◎
	納期（本番時期）	◎	○
	（S/W、H/W の導入時期、テスト、移行時期など）	○	◎
	必要な技術・技術者の資格		◎
	成果物、納入品		◎
	修正プロセスへの参加	○	◎
	共同レビュー	○	◎
	工程計画	○	◎
	開発推進体制	○	◎
	開発手法、開発言語	○	◎
	支援ツール		◎
	ソフトウェア製品の使用		◎
	品質・性能条件		◎
	保守条件		◎
	運用条件	○	◎
④開発体制・開発環境	役割分担	○	◎
	作業場所		◎
	開発機器・使用材料の負担		◎
	貸与物件・資料	○	◎
⑤保証要件	システム	○	◎
	品質保証基準		◎
	システムの性能		◎
	セキュリティ		◎
	デモ・テスト計画		◎
⑥契約事項	発注形態	○	◎
	検収・支払条件		◎
	瑕疵担保責任		◎
	機密保持		◎
	著作権など		◎
⑦その他	用語	○	◎
	外部委託契約に関する管理	○	◎
	リスクに対する相互認識	○	◎
	仕様変更・機能追加などの条件	○	◎

凡例　◎作業主体（&責任あり）　　○作業支援または内容の吟味・確認

SLCP では 44 項目になっているが、ここでは一部詳細化して 47 項目で分類した。

付 7-3　ユーザ部門への教育（T 社事例資料より引用）





	IT部門			利用部門	混成PJ
概要	マネジメントコース	実務者コース	演習付きコース	演習付きコース	出前コースカスタマイズ
対象者	部長・マジメント層	リーダ層	リーダ層中堅	ユーザ	IT部門とユーザ
教育期間	半日	1日	2.5日	2日	半日
開催回数	2	3	2	2	4
受講者数	55	89	29	29	22

８．要求仕様を設計からプログラムまでトレースする方法

要求をトレースすること

ビジネスの関わるソフトウェアでは、最初に要求仕様書の中で実現するべき機能が明確に定義される。次にその機能がコンピュータの中でどのように実行されるかのイメージが、設計作業によって明確にされる。そしてその設計の結果がプログラムの形に変えられて、現実にその機能をコンピュータが実現することになる。

この要求仕様書からプログラム設計書を経由してソース・プログラムまで、ある特定の機能についていえば一本の線が引かれることになる。要求仕様書の中に書かれたものがプログラムから欠落しているとそれは欠陥であり、逆に要求仕様書の中に書かれていないものがプログラムに書かれていると、場合によればそれも欠陥になる。したがって複数の線は全て要求仕様書から始まり、プログラム設計書を経由してソース・プログラムで終わることになる。

ビジネスがコンピュータで処理さえるようになってからも長い間、この「線」で各種の成果物を紐付けしようとする考え方は無かった。今稼働している多くの情報システムには、残念ながらまだこの段階のものが多い。しかしソフトウェア工学の進展と共に、明確にこの線を成果物の中に引こうとする考え方が一般的になり、議論の段階でいえば今は「引くか／引かないか」ではなく、「いかに引くか」に議論の焦点が移ってきている。

もちろんこの線を引くことの大きな目的は、保守作業での負荷軽減である。保守では一般に機能レベルで変更が明確になり、最終的にそれを実現しているプログラムの部分を特定して、そこに必要な修正を施すことになる。

このように見ると、線は一方向で良いように見える。しかし現実の保守の様々な局面やテストの状況を考えると、線は双方向にたどることができる必要である。つまり要求仕様書からプログラム設計書を経由してソース・プログラムに向かう方向だけでなく、ソース・プログラムからプログラム設計書を経由して要求仕様書に至る逆方向の道筋をたどらなければならないこともある。

トレースの方法－１

我々が要求仕様書作成で採用することにした USDM 表記法では、各要求に「要求番号」と呼ぶ固有の ID を付けることになっている。我々も当然、この要求番号の使用を決めた。

したがって単純に考えれば、要求仕様書に記述された要求番号を該当するプログラム設計書にも、その機能を実現しているソース・プログラムにも付けてゆくことで、要求の双方向へのトレースは簡単に実現できることになる。

さらにこの方法を用いることによって開発段階で仕様変更が発生した場合、設計書、あるいはプログラムのどこに影響するかの把握が容易になる。

ただこの方法を採用した場合、設計書にもプログラムの中にも、あちこちにこの要求番号

が明示され、見かけ上たいへん煩雑になるという指摘がある。

トレースの方法－２

しかしこのトレースの方法－1 で示した作業は設計作業固有の性格から、現実はたいへんな手間をかけないと実現できないものかも知れない。つまり設計という作業は淡々と一方向に進捗するものでは決して無く、試行錯誤や行きつ戻りつの繰り返しであることが多い。その過程で要求仕様書に書かれた要求番号の正確な記述をプログラム設計書上に求めることは設計者に過大な負担をかけ、場合によれば実現を危ぶまなければならないようなものかもしれない。このレビューも、たいへんに神経を使うものになるだろう。仮にそうであるとすれば、我々は単純にプログラム設計書やソース・プログラムに「要求番号」を書き込むことは避けて、それに代わるトレーサビリティ実現の方法を考える必要がある。

今ここで提案できるのは、以下の方法である。

要求仕様書に「要求番号」を付けたように、設計でも「設計番号」を付け、プログラムにも対応する番号を、要求番号とは独立にそれぞれ付けておく。そして最後に、要求番号を行に、設計番号、またはプログラムに付けた番号を列に置く複数の表を作成し、トレーサビリティをその表のセルに例えば ○ 印を付けるというような方法で表示をするというものである。この場合はトレーサビリティのために、独立した作業が発生することになる。

この方法は、要求仕様書からシステム設計書、ソース・プログラムへのルートだけではなく、要求仕様書作成の前の作業から要求仕様書へのトレースにも応用することができることに留意しておきたい。

トレースの方法－３

この件について、清水氏から次のようなご提案を頂いた。

要求仕様 （USDM表記法）	設計文書類			プログラム類			
	文書1	文書2	文書3	PG1	PG2	PG3	PG4

図表 8－1　要求仕様のトレースの方法－３

図表 8－1 で、複数列ある「要求仕様」の列に、USDM 形式で要求や要求仕様が書かれる。その要求仕様の列のさらに右側に複数の列を用意し、それらを文書の列、プログラムの列とする。文書の列には、その仕様を扱った設計書などの「章、節」の番号を、プログラムの列には、クラス名や関数名を書き込む。

この表を完成させることで、トレーサビリティを実現することが可能となる。このように USDM 表記法の表を大きくすることで、現実これを紙にプリントすることが困難になるかもしれない。この場合、ユーザ部門に提示する場合などを除いて要求仕様書はプリントし

ない、という決断が必要になる。

作業の仕方として、この表を作業の区切りで埋めてゆく方法がある。設計書を書いている時も、プログラムを書いている時も、そこではどの仕様を扱っているのかということは、多くの場合、設計者やプログラマは意識している。実際、いくつかの関数を書いた後で、その関数単位での動きを確認するが、その際には間違いなく、元になっている仕様、あるいは実現対象の仕様を意識しているので、関数レベルの「単体テスト」を終えるたびに、このマトリクスに関数名を書き込むことができる。というよりも、慣れればそれが作業の「区切り」になる。しかし作業の途中段階でも、状態も表現しようと思えば、表現が可能である。

このマトリクスがあることで、その後に仕様の変更が生じたときに、それが「どの設計書のどこ（章：節）に書かれているのか」「どのソースファイルのどの関数で扱われているのか」がすぐに分かる。また、同じ関数が別の仕様の実現にも関わっていることもすぐに分かる。いわゆる「影響」をチェックしやすくすることができる。

トレースの方法－４

このトレースの方法－４では、要求仕様書や設計書などに特別のことを行わず、トレースの必要が主応じたとき、該当するファイルを全文検索することで必要な部分を発見しようとする方法である。

以上の４つの方法の中では、3 つ目の方法が一番現実的と思われる。これを、現時点では我々の方法としておきたい。

理想的な方法

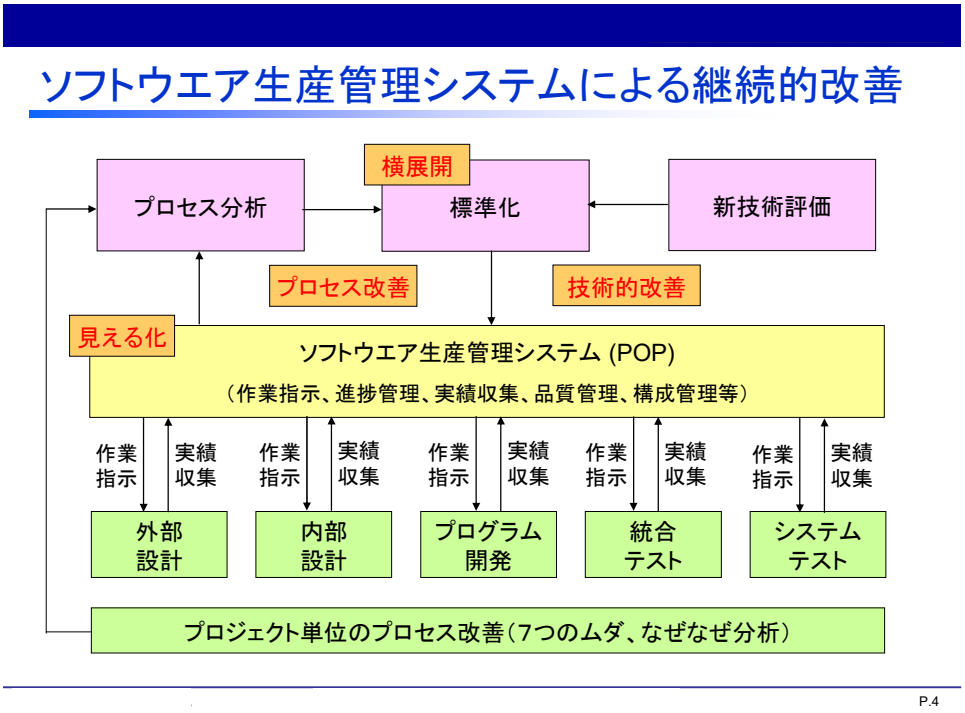
このプロジェクトへの参加企業に中に、情報システムでこのトレーサビリティを実現している企業がある。残念ながら一般の企業がそのような情報システムを持つことはまだ困難と推測するが、ご参考までにそのシステムの概要をここで紹介しておきたい。

その企業（以降「S 社」と呼ぶことにする）は、「情報システムのライフサイクル全体を管理する」ための情報システムを現在開発中で、主要な機能は既に稼働を始めている。このシステムを S 社は「ソフトウェア生産管理システム」と呼んでいる。

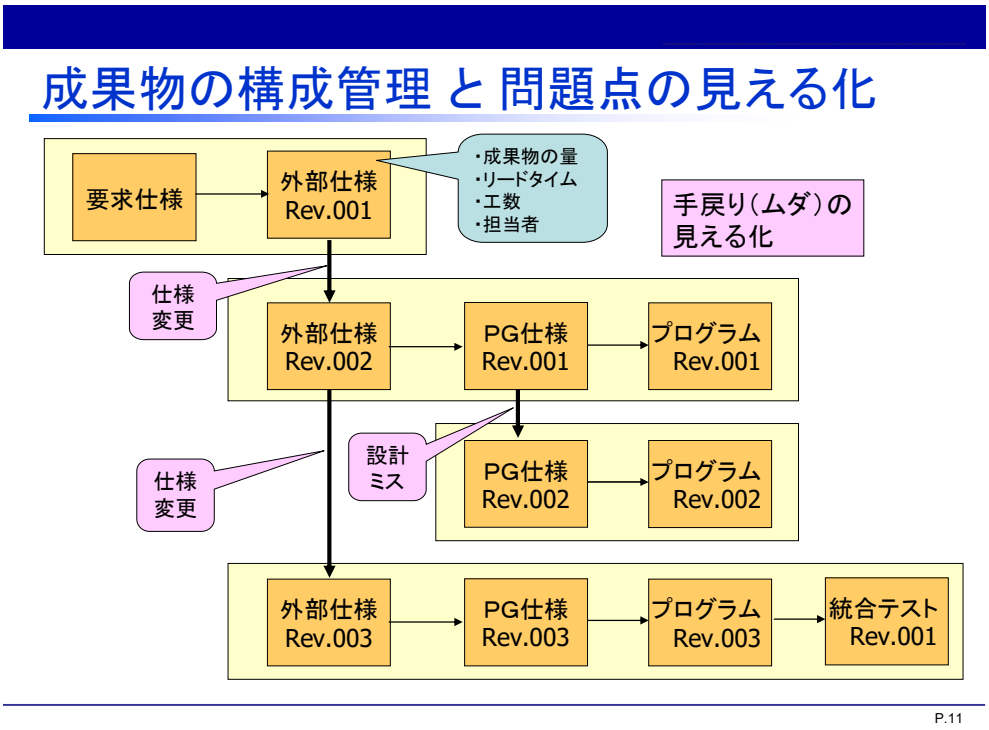
具体的には、外部設計からシステムテストに至るソフトウェアの開発過程で、担当者への作業の指示を行い、作業の実績を可能な限り自動的に収集してそれを進捗管理に結びつけ、品質の管理も行うという情報システムである。

今の日本では、製造業の主な企業にはその製品を生産するための「生産管理システム」が既に稼働している。それと同様今後ソフトウェア会社は、製造業の企業が持っている「生産管理システム」と同じような機能を持つ「ソフトウェア生産管理システム」を持つべきと、S 社の管理者の 1 人は主張する。

このソフトウェア生産管理システムの概要を、図表 8-2 に示す。



図表 8-2　ソフトウェア生産管理システムの構成（S 社事例資料より引用）



図表 8-3　構成品目間の関係（S 社事例資料より引用）

このソフトウェア生産管理システムの中に、ソフトウェアの構成管理の機能が位置づけさ

れている。ただ S 社のソフトウェアの構成管理システムは、ISO がその規格[ISO98a]で定めているような“普通の”構成管理ではなく、「構成品目」の取り扱いに特徴がある。

普通のソフトウェアの構成管理の ISO 版では、その管理の対象物である「構成品目」は、それ自体が全て独立していることになっている。つまり、仮に同じ情報システムの要求仕様書とプログラム設計書、ソース・プログラムなどが全て 1 つのソフトウェアの構成管理システムの中でそれぞれ構成品目（構成管理の対象物）になっていても、それらの間に関連を持っていない。

しかし S 社の場合図表 8-3 に示すように、「トレーサビリティの観点で」これらが全て紐付けされている。そしてその紐付けされた複数の構成品目が、ソフトウェアの構成管理の機能の 1 つである版管理の対象になっている。さらに従来のソフトウェアの構成管理の機能に加えて、仮にある要求仕様書が変更の対象になった場合、この紐付けによって該当するプログラム設計書やソース・プログラムへの変更について、ソフトウェアの構成管理システムがウォーニングを出す仕組みになっている。ここに S 社のシステムの、大きな特徴がある。

構成管理の技術は、勿論ソフトウェアだけに適用されるものではない。多くの業種で、既に適用されている。この場合他の業種には、構成品目間のトレーサビリティを上で述べたような形で既に構成管理のシステムに取り込んでいるところがあるという。別の言い方をすれば、ソフトウェアでまだこのような形になっていないことがあるいは問題なのかも知れない。S 社は、その流れを先取りした先進事例と評価したい。

ソフトウェアの構成管理の目的はソフトウェアに付きものの変更を管理することであり、それぞれの構成品目にバージョン・ナンバーを付けてこの管理を実現している。S 社のような進んだ形のものでなくてもソフトウェアの開発組織は、もし未対応であるならばこのソフトウェアの構成管理の導入を期待したい。

参考文献

[ISO98a] 日本工業標準調査会情報部会審議、「ソフトウェアライフサイクルプロセス－構成管理 TR X 0018：1999 （ISO／IEC TR 15846：1998)」、(財）日本規格協会、平成 11 年 10 月 30 日.

９．データベースの定義方法

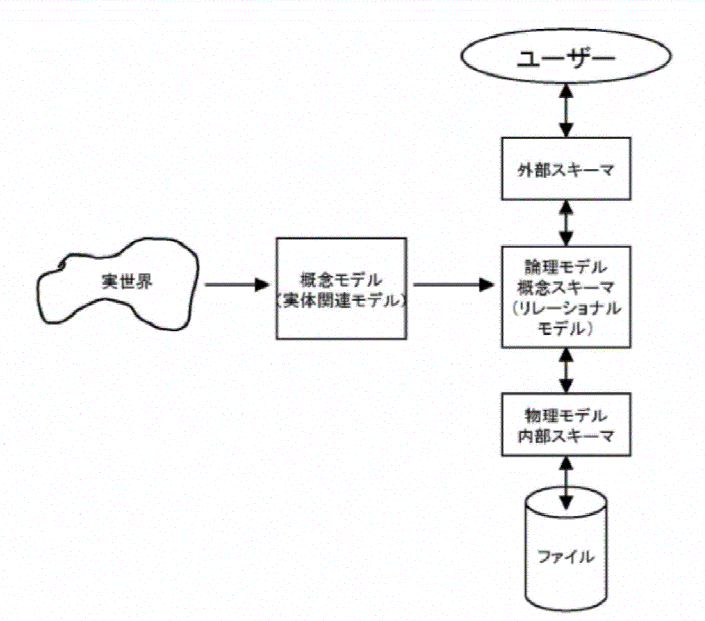
３層スキーマ構造とデータモデル

この世の中の現実をデータベースにどう反映するかという命題について、1970 年代から 80 年代にかけて、世界中で活発な議論が展開された。その結果得られたものの１つが、ANSI（American National Standards Institute）/X3/SPARC がまとめた「三層スキーマ構造」であり、もう１つが「データモデル」の概念である[MAS03]。

三層スキーマとは、概念スキーマ、内部スキーマと外部スキーマを指す。この三層スキーマ構造では、実世界の現実をまず「概念スキーマ」で抽象化し、それを「内部スキーマ」に変換してコンピュータのファイルに落とし込む。さらに「外部スキーマ」を通してユーザに提供するという考え方をとる。三層スキーマ構造では、この３つの層はそれぞれがお互いを制約せず、独自に、自由に決めることができなければならないとしている。この考え方を「三層独立」と呼ぶ。

もう一方のデータモデルは、実世界の現実をまず「概念データモデル」で抽象化し、それを「論理データモデル」の形に展開して、最後は「物理データモデル」でコンピュータのファイルに落とし込むという考え方を取る。物理データモデルと内部スキーマは、基本的に同じものとなる。

ここで紛らわしいのは、一方が概念スキーマと呼び、もう一方が概念データモデルと呼んでいるものが実は全く異なるものであって、三層スキーマ構造の概念スキーマは、データモデルでいう論理データモデルと同じという関係にあることである。この間の関係を、図表 9-1 に示す。



図表 9-1　実世界をデータベースに反映する考え方

データベースはこのように、言葉の上でも考え方の上でも、データベースを構築する技術の裏側にあるデータベース理論も、さらにその背景にある数学の該当する分野も、全てたいへん難しいというのが現実である。

データ中心アプローチとプロセス中心アプローチ

そうは言っても、データベースは現代の情報システムにおいてたいへん重要なものである。今では、データベース抜きでの情報システムは、考えられないという状態になっている。

しかし情報システム化の歴史の中で、最初からそうだったわけではない。ビジネスに関わる情報システムでは、出力が常にたいへんに重要なものだった。これは、今でも変わらない。この情報システム化の最初の段階は、その重要な出力を作るためにコンピュータが何を行うべきかという命題を考え、その結論に基づいて情報システム作りが行われていた。この情報システムの作り方は、後で「プロセス中心のアプローチ」と呼ばれるようになった。

プロセス中心のアプローチには、データベースという概念はない。むしろあるのはファイルであって、そのファイルをプログラムがバケツリレーをするように順次つないでいって、最後に欲しい出力を作るという形を取る。このようなファイルは他のアプリケーションと共有することが難しく、そのため情報システムからの出力で整合性がとれない、あるいは内容の不一致が起きるというような問題が常時発生していた。

ソフトウェアの進歩がリレーショナル・データベースのデータベース管理システム（DBMS）の機能を充実させ、ハードウェアの進歩がやはりリレーショナル・データベースの処理の効率を向上させて、データベースが実用的なものになると共に、情報システムの作り方でも新たな方法が提唱された。ビジネスの現在の状況をデータベースの中に的確に格納して、情報システムの中心に位置づけようとする考え方である。つまり、必要な出力を作成するためのデータをデータベースに格納しておく。それを更新するべきトランザクションが発生すると、即データベースを更新する。出力が必要になると、その更新済みのデータベースから必要な出力を作成する、という考え方と言い換えても良い。この考え方を「データ中心アプローチ」と呼ぶ。

データ中心アプローチでは、最初にデータベースの概念設計を行う。つまりこれから開発しようとする情報システムで核になるデータベースを、一番始めに明確にしようとする訳である。

実体関連図

このデータベースの概念設計の結果作られるものを、概念データモデルと呼ぶ。概念データモデルは、今のところ実体関連図（英語では Entity Relationship Diagram、省略して ERD、あるいは ER 図と表されることがある）、あるいはクラス図で表される。

この報告書では実体関連図を用いて議論し、以下でこの実体関連図の作り方の概要を述べる。ただ以下で述べるものは大学で使う教科書[MAS03]を基にした極めて一般的な入門編であり、実際の情報システム用の概念データモデルを構築するためには、もっとしっかりした方法論に基づく手法の採用を推奨する。今の日本には PLAN-DB[TUB05]や T 字型 ER 手法[SAT05]など、優れた概念データモデル作成の方法論が普及している。これらがその場合の候補になりうるものである。

しかし方法論が異なれば、実体関連図の表現方法が変わり、あるいはすぐ後で述べるデータベース化の対象物の呼び方も異なる。ここに、注意が必要である。

データベース化の対象物

概念データモデルの作成で最初に行うべき作業は、データベース化の対象物を明確にし、実体関連図を描くことである。これには、2 つの方法がある。トップダウンで行う方法と、ボトムアップで行う方法である。

トップダウンでこれを行うには、充分な経験を積んだスーパーSE が必要になる。彼／彼女は仕様書に目を通しただけで、実体関連図を描くことができ、データベース化の対象物を明確にできる。この方法だと時間は少なくてすむが、スーパーSE がいなければ実施できないという難点がある。それに対してボトムアップの方法は、時間はかかるけれど基本的には誰でも行うことができるという利点がある。

ボトムアップの方法によるデータベース化の対象物を明確にする作業は、これから開発しようとする情報システムの出力を分析することで、行うことができる。データベースとは「必要な出力を作るためのデータを格納したもの」であるから、その情報システムで何を出力するべきかを明確にすると、必然的にデータベースに格納するものが明確になる。別の言い方をすると、出力する必要のないデータは、データベースに格納する必要がない。これは、自明のことである。データベースに格納するべきものは、この自明のことを裏返しにした作業を行うことによって、明らかにすることができる。

まだ手作業でも行われたことがない新たな業務を情報システム化する場合でも、その情報システムで作成しなければならない出力を明確にすることができれば、以下で述べる方法を採用することができる。

ここでいう出力には、画面と帳表の他に、他のアプリケーションとのインタフェースになるファイルや、これからの情報システムでは、他の情報システムやユーザにネットワーク経由で送るメッセージも対象にすることが必要である。

出力の種類とそれを構成しているデータ項目が確定すると、そのデータ項目を全部名寄せする。つまり全部の出力からデータ項目だけを集めて、重複しているものは 1 つだけ残して後は消去する。データの標準化（「1 つのデータ項目に 1 つの名前」を実現すること）が実施されていると、この名寄せでは表面的に名前だけを見て消去すればよい。データの標準化が実施されてなければ、名前だけを見て単純に消すことができないため、消すに当た

って注意が必要である。いずれにしろここでは、重複と漏れのないデータ項目の集合が求まればよい。

次にこのデータ項目を 1 つずつ、何に属するデータ項目であるのかを丹念に問いかけて、データ毎にグループを作成する。例えば「顧客氏名」は顧客に属するデータ項目であり、「従業員の住所」は従業員に属するデータ項目である。「商品単価」が商品毎に付けられた標準単価であれば商品に属するデータ項目であり、「取引単価」が標準単価を参考にして決められる取引毎の単価であるなら取引に属するデータ項目である。この段階で、対象業務をよく知っているユーザの参画が必要になる。

このようにして作られたデータ項目の集まりであるデータが、データベース化の対象物である。ここで、そのデータに名前を付ける。あるいは改めて名前を付けると言わなくても、それを意識してこれまでグループ分けを行ってきたのかも知れない。

ここでこの名前には、2 つの種類がある。純粋の名詞（「する」を付けても動詞にならないもの。例えば、顧客、商品など。）と、動詞の語幹としての名詞（「する」を付けると動詞になるもの。例えば、取引、購入など。）である。以降で純粋の名詞を「実体」、動詞の語幹を「関連」と呼ぶ。

正規化について

更新が入るデータベースでは、そのデータベースに格納するものには「正規化」と呼ばれる作業を行わなければならない[MAS03]。これはデータベースを更新するに当たって、更新の対象ではないデータを消去してしまったり、変更ししてしまったりする「更新異常」と呼ばれる事態を引き起こさないようにするためである。

ある概念データモデル作成の方法論提唱者は、「正規化によって“One fact, One Place”が実現される」と表現している。正規化を正しく行うと、まさに「データベースの 1 つの場所には 1 つの事実だけが格納され、逆に 1 つの事実は 1 つの場所にしか格納されていない」ということを実現することができる。ただしパフォーマンスなどを考慮して、意識してデータベースの複製（レプリカ）を持つ場合は別の話になる。

我々が普通に使うデータベースでは、正規化は「第 3 正規形」と呼ばれるところまで行えば良いとされている[MAS03]。しかし次の「実体関連図の描き方」で述べるように作業すると、第 3 正規形の次に高いレベルの「ボイス・コッドの正規形」のレベルで正規化を行うことができる。

このような方法で、データの冗長性が無い、コンパクトなデータベースを作ることができる。ただし教科書に書いてある正規化の方法で作業を行うと、ユーザにとってなじみがある「情報」がバラバラの「データ」にばらされ、分かりにくい、難しいものに変わってしまうという印象が持たれる可能性がある[MAS03]。

このようなこともあって、データ・ウェアハウスなど更新を伴わないデータベースの場合は、そもそも「更新異常」が起きるはずがないと言うことから、正規化の作業は必要とし

ない。

実体関連図の描き方

データベース化の対象物を決めたとき、その対象物を名前が純粋な名詞で表されるもの、つまり「実体」と、動詞の語幹で表されるもの、つまり「関連」に二分した。もう一度ここで、名詞で表される実体を「人など」に関連するもの（例えば、顧客、学生など。）と「ものなど」に関連するそれ以外のもの（例えば、商品、科目など。）に分ける。結果としてデータベース化の対象物は、次の 3 つに分けられたことになる。

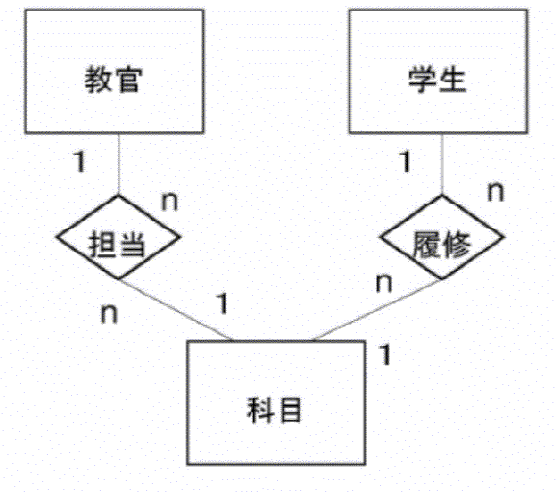
- 人などに関わるもの
- 動詞の語幹
- ものなどに関わるもの

ここで、人などに関わるものを主語、動詞を述語、ものなどを目的語にして、全ての組み合わせで文章を作ってみる。その結果はその業務で、あるものは正しく、あるものは正しくない、という形になる。その正しいものだけを残しておき、実体関連図に記述する。

実体関連図では、実体（名詞）は四角、関連（動詞）は菱形で表し、その間を線で結ぶ。したがって 1 つの文章は、主語と目的語の 2 つの四角（実体）と述語の 1 つの菱形（関連）が一本の線で結ばれた形になる。何度も同じ実体が図の上に表れるとき、それらをまとめて、1 つにしても良い。

その線の実体と関連に近いところに、多重度を記す。多重度は、普通は実体側が 1 、関連側が n （1 以上）になることが多いが、時には関連側も 1 になることもある。

このようにして、2 つの実体の間に必ず関連を挟む形で概念データモデルを作ると、そのモデルは第 3 正規形よりもう一つレベルの高い「ボイス・コッドの正規形」になる[MAS03]。



図表 9-2　実体関連図の例

実体関連図のシンプルな例を、図表 9-2 に示す。

キィについて

ここで、全ての実体と関連を構成するものをそれぞれ唯一無二に識別できるデータ項目を探す。探して見つからなければ、そのようなデータ項目を作ることがある。このデータを唯一無二に識別できるようにするデータ項目を「キィ」という。

実体の場合は、キィ専用のデータ項目を持つのが普通である。例えば、顧客には顧客コードを用意してそれをキィにする、商品の場合は商品コードを持つ、という形にする。既にこのようなデータ項目が用意されているのが普通だが、仮に用意されてない場合は、SE が自分で作る必要がある。

関連は 2 つの実体に挟まれているので、その 2 つの実体のキィ項目の組み合わせをまずキィとする。例えば、「顧客が商品を購入する」というというケースでの「購入」のトランザクションのキィは、まず顧客コードと商品コードの組み合わせで考える。その組み合わせだけでそのトランザクションが唯一無二に識別できるのなら、キィはこれに決めればよい。

しかし、これでは決まらないのが普通である。これで決まるケースとは、その顧客が彼（彼女）の人生の間で、その商品を購入することは常にたかだか一回しかないことを意味する。普通は、そのようなことはあり得ない。このような場合には、何かのデータをこれに付加して「唯一無二」を実現する。例えば、同じものを一日に二回以上買うケースが全くない場合には、購入年月日をキィの 3 つ目の項目にすればよいかも知れない。一日に何回も買うケースがある場合はどうするか。それを決めるのが、ある意味での SE の腕の見せ所になる。

このようにして、概念データモデルを完成させることができる。これが要求仕様書に次ぐ、要求仕様段階での 2 つ目の成果物になる。

インフォメーション・エンジニアリングからエンタープライズ・アーキテクチャへ

この報告書の目的は、単独の情報システムの開発において要求仕様書をどう書くかに答えを出すことである。その意味で、会社全体のデータモデルを考えるという話は、本来の目的から外れる話になる。それを承知の上で、全社データモデルとエンタープライズ・アーキテクチャの重要性について、ふれておきたい。

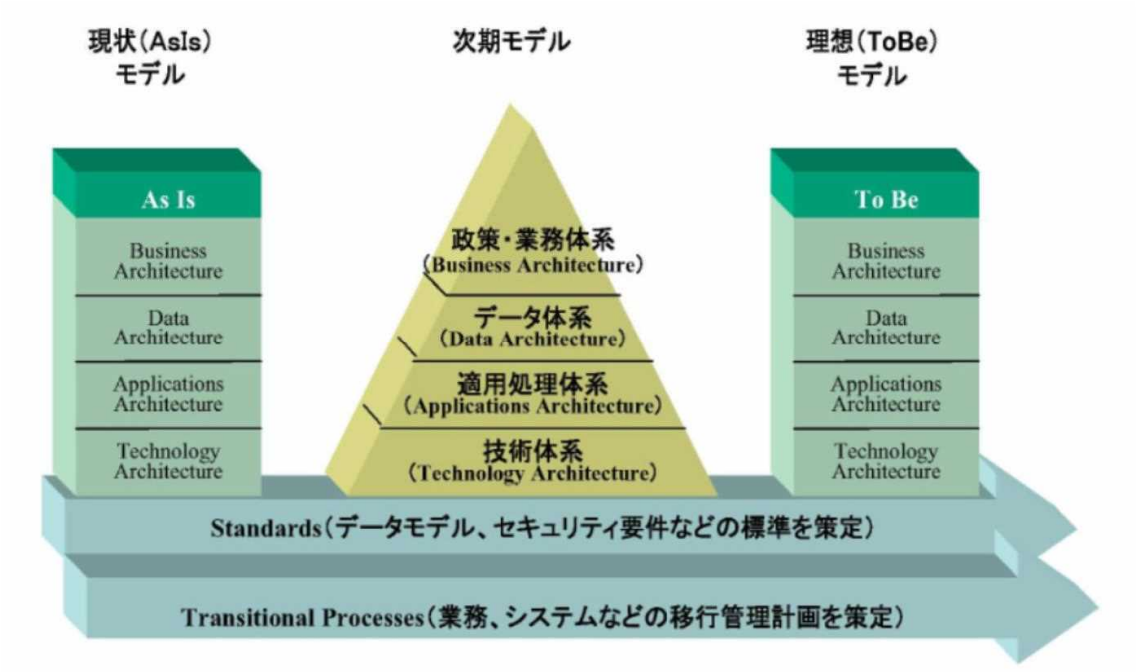
1980 年代後半、「全ての企業は、まずその会社の全社データモデルを構築し、それを基にして必要な機能を順次開発するべし」という考え方が、アメリカを中心に広がったことがある。この考え方を、当時は「インフォメーション・エンジニアリング（以降は省略して IE と記す）」と呼んだ（[MAR89] [MAR90a] [MAR90b]）。IE は日本ではさほど多くの企業に受け入れられることなく、そのうちに話題になることが少なくなった。

そして今は、単にデータだけを考えるのではなく、技術、アプリケーション、そしてビジネスまで全体を統合した形で、企業の情報システムの体系を考えようとする議論が行われ

ている。それを「エンタープライズ・アーキテクチャ（以降は省略して EA と記す）」と呼ぶ[MET06a]。この EA の体系を、図表 9-3 に示す。

この研究会の会員企業に、T 字型 ER モデルに則って全社のビジネスモデルを構築し、それをベースにして、UDSM 表記法も使用して要求仕様書を作り、併せて実体関連図で概念データモデルを作成して、それらに基づいて個々の情報システムの開発を行っているところがある。

またこの研究会の別の会員企業に、数年前に大規模な合併をして、その結果生まれた会社がある。合併前の会社はそれぞれの業界で既に重要なプレーヤであり、それぞれの情報システムを持っていた。合併後この新しい会社の IT 部門がまず行ったことは、EA をベースにした新しい会社の全社モデルを作ることだった。そしてそのモデルを完成させた後、それを基に必要な機能を充足する情報システムを順次開発し、それを完成させて企業としての実質的な合併を実現した。



図表 9-3　EA の体系（「経済産業省のホームページ」より）

我々は、企業にとって効果的な情報システムを開発し、維持し続けるために、IE や EA が要求しているような本格的で抜本的なアプローチを、少なくとも一度は真剣になすべきなのかも知れない。

参考文献

[MAR89] ジェームズ・マーチン著、竹林則彦監修、三菱 CC 研究会 IE タスクフォース

訳、「インフォメーション・エンジニアリングⅠ　統合化 CASE のための方法論」、(株)トッパン、1991 年.

[MAR90a] ジェームズ・マーチン著、竹林則彦監修、三菱 CC 研究会 IE タスクフォース訳、「インフォメーション・エンジニアリングⅡ 統合化 CASE による計画と分析」、(株)トッパン、1992 年.

[MAR90b] ジェームズ・マーチン著、竹林則彦監修、三菱 CC 研究会 IE タスクフォース訳、「インフォメーション・エンジニアリングⅢ 統合化 CASE による設計と製作」、(株)トッパン、1994 年.

[MAS03] 増永良文著、「リレーショナルデータベース入門［新訂版］－データモデル・SQL・管理システム－」、Information & Computing – 43、サイエンス社、2003 年.

[MET06a] 「EA ポータル」は、次の URL からダウンロードできる。
http://www.meti.go.jp/policy/it_policy/ea/index.html

[SAT05] 佐藤正美著、「データベース設計論－T 字型 ER 関係モデルとオブジェクト指向の統合をめざして」、ソフト・リサーチ・センター、2005 年.

[TUB05] 椿正明著、「名人椿正明が教えるデータモデリングの技 データ中心システム開発原論」、翔泳社、2005 年.

10. 画面／帳表のデザインとユーザビリティ

画面／帳表のデザインの重要性

インターネットの普及・拡大と共に、画面の操作性機能が一段と向上した。以前の汎用機の情報システムでは実現できなかった以下のような機能を、今では縦横に駆使することができるようになっている。

- 画面上の文字の大きさやフォントの種類が、ある範囲で自由に選べるようになった。
- カラー画面が普通になった。
- ラジオボタンなど、画面をコントロールする機能が充実してきた
- イメージデータや写真なども活用できるようになった

しかし画面や帳表の設計方法は以前の汎用機時代のままの状態で留まっており、画面などの設計技術と画面で実現できるものの間の乖離が広がっている。

ユーザはインターネットの画面になじんできており、画面の操作性などについての要求が高くなってきている。しかし情報システムを作る側はユーザ・インタフェースについて、これまでと同様さほど時間も気持ちも使わない状態が続いている。このためテスト工程の最後でユーザが実際に画面の操作を行ってみて改善要求を多く寄せたり、カットオーバ以降稼働を始めてから手直しの要求を寄せたりする事態が多くなっている。

日本情報システム・ユーザー協会（JUAS）ではこのような事態を避けたいとして、2005年度より「ユーザビリティ研究会」をスタートさせ、研究活動を行ってきた。この章では、この研究会のこれまでの研究成果を紹介したい。なおこの分野の詳細については、別冊の2006年度版「ユーザビリティ研究会」の報告書を参照していただきたい[JUA07]。

ユーザビリティとは何か

最初に、この章の標題の後半にある「ユーザビリティ」という言葉の意味から考えてみたい。一言でいえばユーザビリティとは、「使いやすさ」を表す言葉と捉えたい。

製品の使い勝手には、ユーティリティとユーザビリティの2つの側面がある。ユーティリティとは「機能／性能」のことであり、使い手にとって製品のプラス面がどれだけ高いかを表す言葉と定義できる。一方のユーザビリティは、「使いにくさ」、「分かりにくさ」などのマイナス面がどれくらい少ないかを表す言葉と定義できる [IID03a]。

これまで新製品の開発では、いかに高いユーティリティを持たせるかに多くの力が注がれてきた。その結果ユーティリティは非常に高いけれど、その機能・性能を充分に使えない、使い切れないという製品があふれることになってしまった。そこで今ユーティリティの高さと共に、高いユーザビリティを持つ製品が求められるようになってきている。

ユーザビリティは、公式にISOの規格（ISO 9241-11：1998）で定義されている。その定義に基づけば、ユーザビリティとは以下のものである[ISO98b]。

- 使用性（Usability）：ある製品が、指定された使用者によって、指定された利用の状況下で、指定された目的を達成するために用いられる際の、有効さ、効率及びユーザの満足度合い。
- 有効さ（Effectiveness）：ユーザが、指定された目標を達成する上での正確さ及び完全さ。
- 効率（Efficiency）：ユーザが、目標を達成する際に正確さと完全さに関連して費やした資源。
- 満足度（Satisfaction）：不快なことがないこと、及び製品の使用に対しての肯定的な態度。

ユーザビリティを「使いやすさ」と考えると、この定義の中でそれに近い言葉は「満足度」だけのような気がして、この定義にはやや違和感を覚えるかも知れない。しかしいくら使いやすさの面で良いものを持っていようと、本来我々が必要としている機能をこの情報システムが充分に持っていないのであれば、それは使いやすさ以前の問題である。したがって我々はこの定義を、素直に受け入れる必要がある。

なおこの ISO の規格を JIS 化するに当たって、ユーザビリティには「使用性」の言葉が当てられている。しかしこの「使用性」という言葉は日本語としてなじみがないので、最近はカタカナで「ユーザビリティ」と書かれることが多いようである。この報告書でも、このカタカナ表示を用いることにする。

ユーザ・インタフェースのユーザビリティ

ユーザビリティの定義を情報システムのユーザ・インタフェースに当てはめると、ユーザ・インタフェースは次の 5 つの構成要素を持つと、ウェブ・ユーザビリティの権威である Jakob Nielsen 博士は述べている[IID03b]。

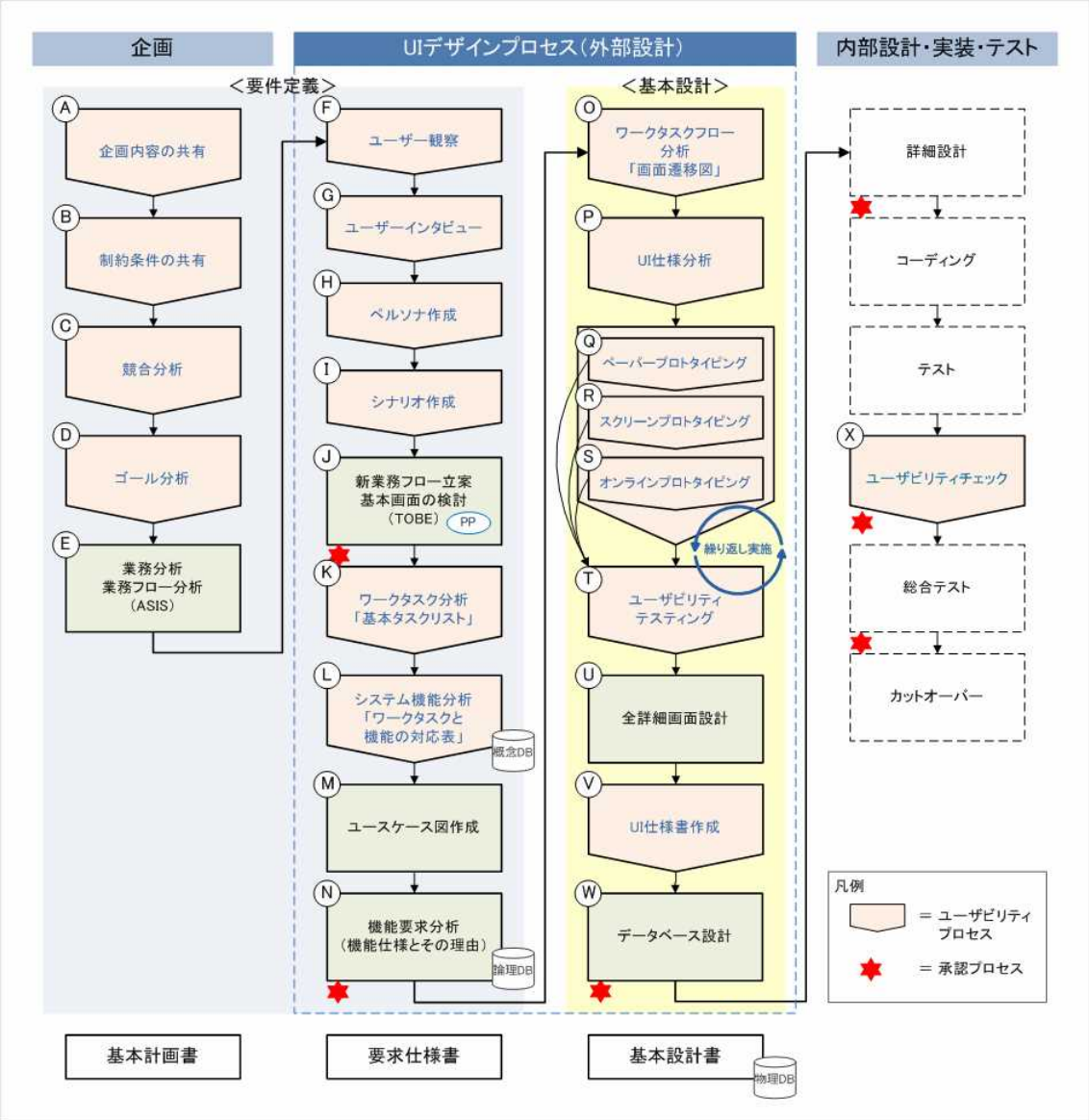
- 学習しやすさ：システムは、ユーザがそれをすぐ使い始められるよう、簡単に学習できるようにしなければならない。
- 効率性：一度学習すれば、あとは高い生産性を上げられるよう、効率的に使用できるものでなければならない。
- 記憶しやすさ：ユーザがしばらく使わなくても、また使うときにすぐ使えるよう覚えやすくしなければならない。
- エラー：エラーの発生率を低くし、エラーが起こっても回復できるようにし、かつ致命的なエラーは起こってはならない。
- 主観的満足度：ユーザが個人的に満足できるよう、また好きになるよう、楽しく利用できなければならない。

ただしここで述べたユーザビリティ特性は全てのユーザに一律に当てはまるものではなく、情報システムが変わったりユーザが変わったりすることで、対象となる特性が異なることがあり得る。

またユーザ・インタフェースのユーザビリティについての日本の権威者である黒須正明教授によれば、ユーザビリティには操作性、認知性、快適性の 3 つの側面があって、それぞれが人間工学、認知工学、感性工学の 3 つの学問領域と関係を持っているという[IID03b]。

ユーザ中心設計（UCD）

ユーザビリティの優れた情報システムを構築するための手法をユーザ中心設計（User Centered Design : UCD）という。JUAS のユーザビリティ研究会が開発した UCD の作業手順（JUAS-UCD モデル）を、図表 10-1 に示す。



図表 10-1　JUAS-UCD モデルの全体像（[JUA07b]より）

この企画段階で行うべきとしている事項は、これまでの情報システムの開発で行うべきことと、大きな差はない。違うところがあるとすれば、「この情報システムではユーザビリティを重視する、つまり設計に UCD の要素を取り入れる」ということを宣言し、関係者一同の合意を取り付けることであろう。

設計段階から、UCD 独特の考え方で作業を行う必要がある。ただしここでいう設計作業は、プログラムやデータベース、あるいはユーザ・インタフェースなどの設計の前に行うべきとしている。場合によれば、アーキテクチャの設計前に行う必要がある。後述するが、UCD ではユーザへの出力を考えることが最優先であり、この設計段階で想定したものを現実のものとしてユーザに提供するために必要なアーキテクチャを、アーキテクチャの設計段階で行うべきという考え方があるからである。

これまでの情報システムの開発ではまず処理するべき内容から処理の方式を固め、それを実現するためにアーキテクチャを決めていた。その上でそれをベースにしてプログラムやデータベース、ユーザ・インタフェースなどを決定していた。UCD では、この順序が一部変更になる。そしてこの一連の作業の成果物は、「UI 仕様書」である。

画面デザインプロセス

画面デザインプロセスの最初の一連の作業は、ユーザ観察とユーザ・インタビューである。この 2 つの作業を通して、この情報システムのユーザについて、ユーザ像とその作業目的、作業内容（UCD ではユーザが行う作業を「タスク」と呼ぶ)）を明確に認識する。UCD ではあくまでユーザが中心であり、その人たちがこの情報システムを使用して充分な成果を挙げることが肝要であるから、この 2 つの作業はたいへんに重要なものとなる。

この結果を踏まえて、UCD では、ペルソナとシナリオを作成し、タスクの分析を行う。

ペルソナとは、この情報システムの設計に当たって想定する「架空のユーザ像」である。それを一般論ではなく、非常に現実的に定めるところに特徴がある。例えば氏名、職務、性別、年齢、学歴、職歴、性格、趣味、現住所、家族構成、特技などを明確に定める。UCD とは、この人が使いやすいと考える画面／帳表を明確に設計すること、といっても良い。ユーザが複数の種類存在する場合は、複数のペルソナを想定することになる。

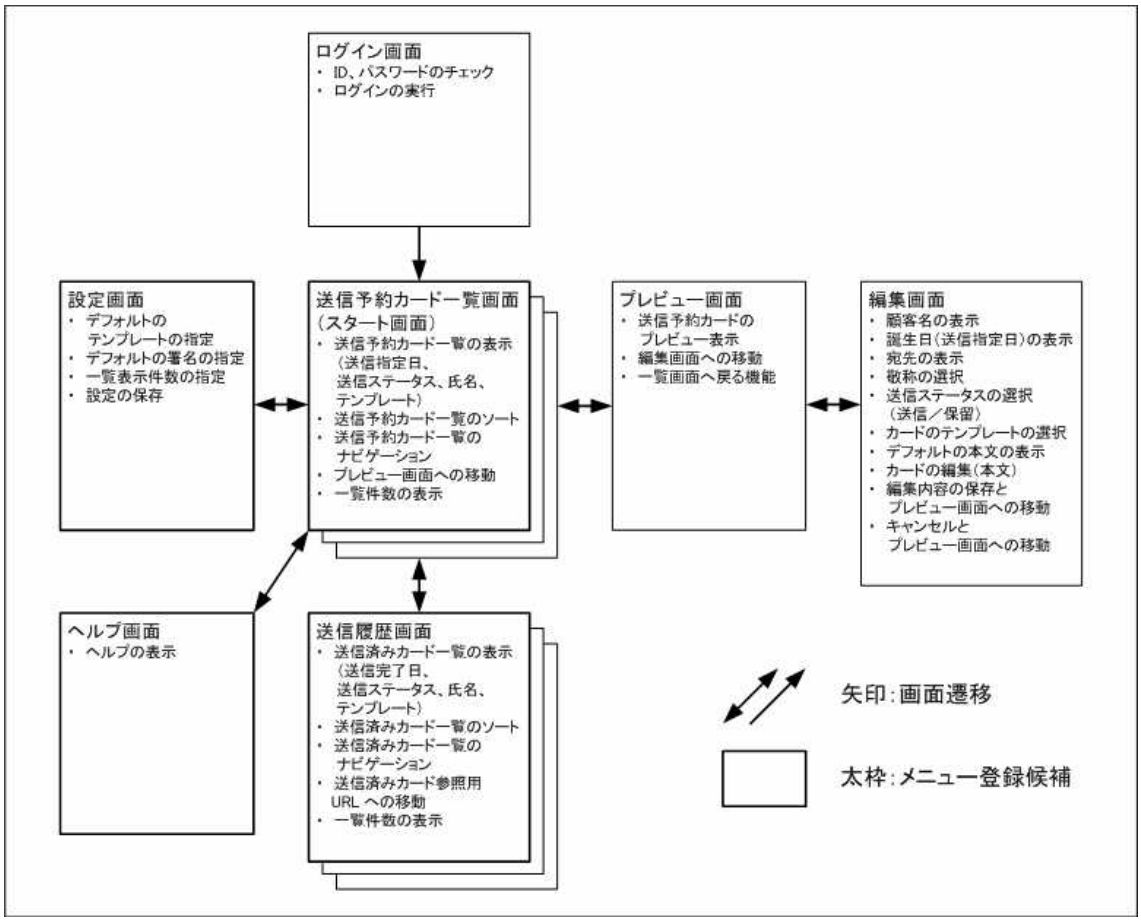
ペルソナが定められると、その人が仕事を遂行するためにこの情報システムをどのように使うのかを、シナリオを作成して明確にする。その「シナリオ」の中で、動詞で表されている作業をタスクとして洗い出し、次の作業であるタスク分析を行う。シナリオに記載されたもの以外にもペルソナが行う作業があれば、それらもタスクとして分析の対象にする。タスク分析で、ペルソナがその作業を遂行するために「タスク」として何の作業を行う必要があるのかを漏れなく明らかにする。そしてその結果を、「基本タスクリスト」にまとめる。

次に、このタスクをペルソナが行うことができるようにするために、情報システムはどんな機能を持つべきかを挙げる。この作業が「機能分析」である。ここで必要な機能が明確

になると、それを我々の要求仕様書に取り込むということが、UCD での本格的な取り組み方である。

プロトタイピング

ユーザが行うべき作業がタスクという形で明確になり、そのための情報システムが持つべき機能までが明確になると、ユーザビリティの観点からの次の作業は「画面の流れ」を明確にすることになる。これを UCD では、「タスクフローの作成」と呼んでいる。個々の画面は 1 つの小さな機能を実行するものであって、その複数の画面の組み合わせ（流れ）で、もっと大きな機能を実現することができる。これをタスクフロー・ダイアグラムの形で表現する。この段階ではまだ、個々の画面のデザインまでは行わない。それぞれの画面を使って、何を行うことができるのかが明確になっていればよい。タスクフロー・ダイアグラムの例を、図表 10-2 に示す。



図表 10-2　タスクフロー・ダイアグラムの例（[JUA07a]より）

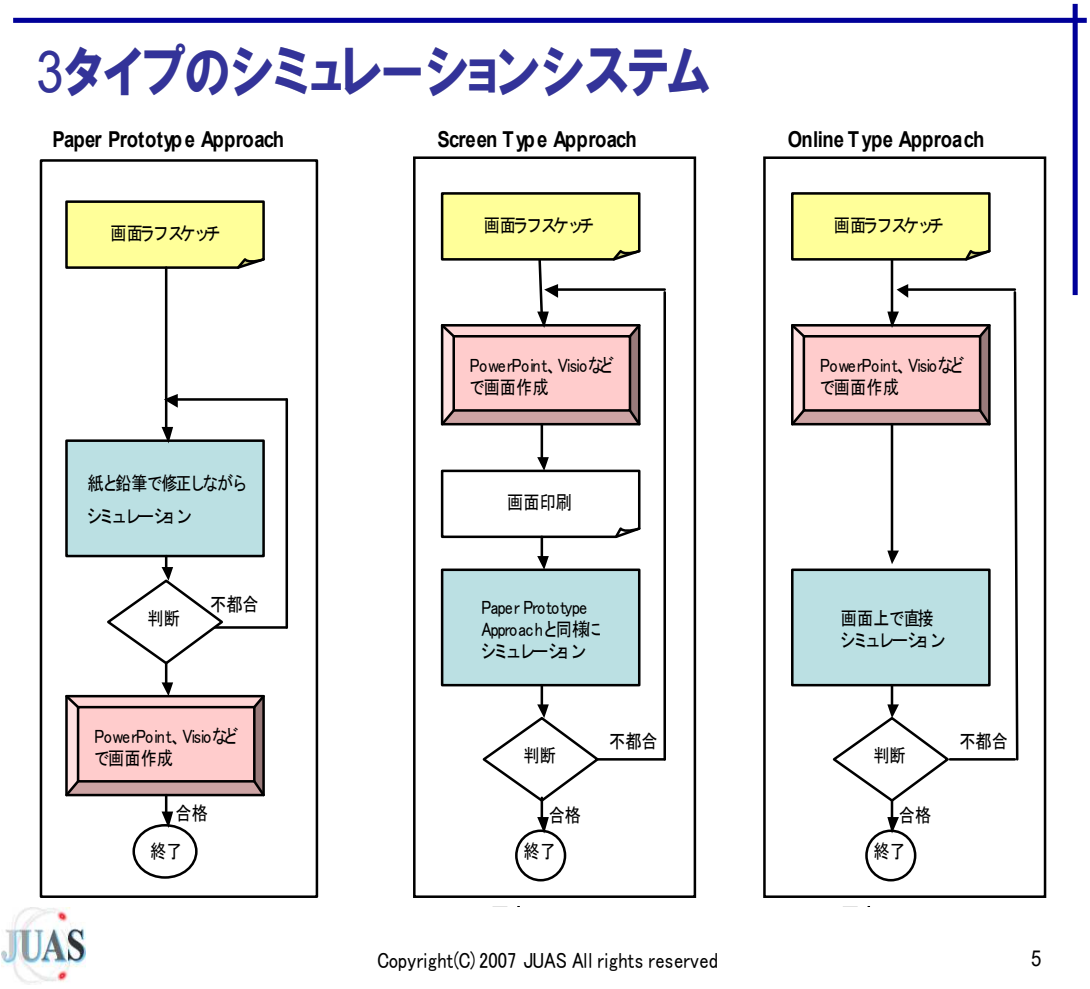
タスクフローが明確になると、次はいよいよ個々の画面のデザインを行う段階になる。個々の画面のデザインはプロトタイプを作成し、ユーザと確認を繰り返しながら固めて

ゆくという方法を取る。ここでのプロトタイプ作成は、最初は「ペーパー・プロトタイピング」と呼ばれる、画面のイメージを鉛筆などで紙の上に書いたもので行うのがよいという。ペーパー・プロトタイピング・アプローチは、ドイツのキャロライン先生が提唱された手法である。紙の上に「鉛筆で定規も使わずにラフに画面レイアウトを書き、その画面を基にしてシミュレーションをする方法」である。訂正は消しゴムを使い、鉛筆で書き直すだけであるから迅速・簡単である点に特色がある。

これを何度か繰り返してユーザの嗜好が固まると、次は「スクリーン・プロトタイピング」に移る。これは PowerPoint や Visio などを使って行うプロトタイピングである。最後に「オンライン・プロトタイピング」によって、ユーザとのインタラクションを実現する。これも PowerPoint などスクリーン・プロトタイピングで使ったツールを使用して行うが、リンク機能などを使用して前述のユーザとのインタラクションを実現するところに特徴がある、

この 3 つの方法を併用しながら、ユーザの画面の嗜好を把握する。ここで両方の方法でプロトタイプを作成する画面は、全体の画面の 5～10%程度で良いという。

プロトタイピングの 3 つの種類のシミュレーションの比較を、図表 10-3 に示す。



図表 10-3　プロトタイピングの 3 タイプのシミュレーション

プロトタイピングのシミュレーションには、次の 4 人の役割が必要である。

① ユーザ：データ入力をする人。

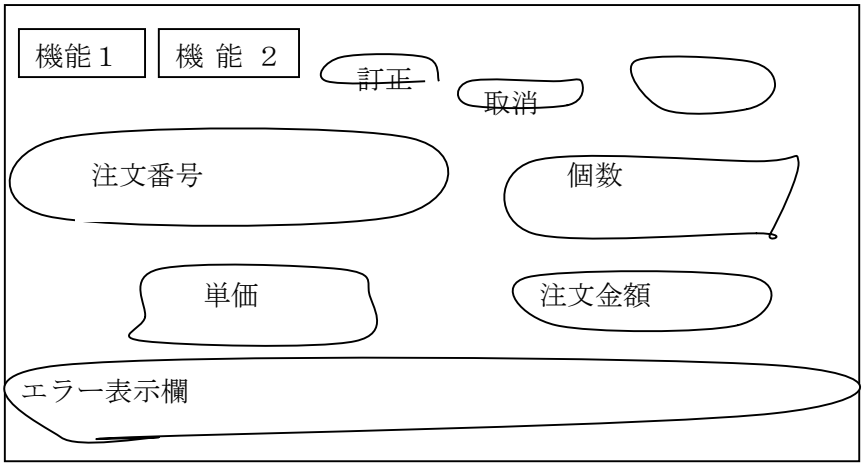
② コンピュータ役：紙画面に記入されたデータが正しいのか、間違っているのかといった、コンピュータが内部で判断する機能を持ち結果はアクションで示す。原則として「言葉を一切話してはいけない」制約を課せられている。

③ 進行役：何のデータを入れるのか、次はどのようなアクションを要求するのかを、テスト計画書に則り指示する役目を持つ。

④ 観察者：進行役の指示に基づき入力者が操作する状況を観察し「ここは何をしたら良いのか、入力者が何秒間程度悩んでいる。改善する必要がある」などと観察結果を記録し、画面設計の訂正をアドバイスする役目を持つ。

③と④を同一人が受け持つ場合もある。

例えば、図表 10-4 で示す次の画面をシミュレーションしてみる。



図表 10-4　シミュレーションで想定した画面

図表 10-5　シミュレーションのイメージ

手順	分担	説明
1	ユーザ	ユーザはこの紙画面の前に座る。
2	進行役	進行役はテストの目的、方法の概要を説明する。
3	進行役	「では始めましょうか。最初に注文番号を入れてください。」
4	ユーザ	「えーと、ここに書けば良いのだな」、とつぶやきながら、注文番号を鉛筆で書く（桁数、英数字の区別は教えられてある）。
5	計算機役	OK なので、黙って紙を上げ下げする。
5	進行役	100 個注文を入れてください。
6	ユーザ	個数欄に鉛筆で 100 と書く。

7	計算機役	OK なので、黙って紙を上げ下げする。
8	進行役	間違えました。数量を 10 個にしてください。
9	ユーザ	どうして良いのか？考え込んでしまう。
10	観察者	何分考えたかを計測しておく。
11	進行役	「分らなければ次に行きましょう」と次の入力作業に移る。
12		以下テスト計画書に従い順次テストのシミュレーションをつなげる。

「ペーパー・プロトタイプ・アプローチ」と「スクリーン・プロトタイプ・アプローチ」、「オンライン・プロトタイプ･アプローチ」の３種類を使い分けることが重要である。ここでこの３手法の特徴を整理すると、図表 10-6 のようになる。

図表 10-6　プロトタイプアプローチ

比較項目	ペーパー・プロトタイプ・アプローチ	スクリーン・プロトタイプ・アプローチ	オンライン・プロトタイプ･アプローチ
画面設計	鉛筆で手書き。 修正が極めて簡単であるが、シミュレーション完了後、SE やプログラマに伝えるためにパワーポイントなどのツールを活用して画面情報の入力を行う。	PowerPoint または Visio などを使用して最初から描く。 若い SE には手書きと比較しての負担感は低い。	PowerPoint または Visio などを使用して最初から描く（本番で活用する画面を、VB などを用いて作成するのではないことに注意)。 若い SE には手書きと比較しての負担感は低い。
シミュレーションの方法	入力者、コンピュータ役、進行役、観察記録者が一定の手順で行う。	役割は左に同じ。	役割は左に同じ。 ただし PowerPoint 等の操作に堪能なコンピュータ役が必要である。
画面の訂正	消しゴムと鉛筆で修正（シミュレーション結果を基にパワーポイントなどで再入力が必要)。	PowerPoint などで訂正する。 （やや煩雑ではあるが画面訂正の手間は大きなものではない)。	PowerPoint の画面を直接に変更する。 （本番で活用する VB などの画面ではないことに注意)。
画面間の結合	なし。	なし。	画面リンク機能の活用が可能。

類似画面の制作	手書きで書き直すので多少わずらわしい。複写機を活用することは可能。	コピー機能が使えるので簡単。	コピー機能が使えるので簡単。
小規模システムへの適用	可能（負荷増感はない)。	同左。	同左。
大規模システムへの適用	冗長であり負担感を伴う。	コピー機能を活用すれば、負担感少なし。	コピー機能を活用すれば、負担感少なし。
将来性	○	◎パソコン操作に強い方には向いている。	◎パソコン操作に強い方には向いている。

「ペーパー・プロトタイプ・アプローチ」を最初の数画面について試み、入力コンセプトや入力イメージを固めたあと、「スクリーン・プロトタイプ・アプローチ」や「オンライン・プロトタイプ・アプローチ」を活用する、などの使い分け方もある。いずれにしても、関係者を集めてのシミュレーションを実施する手順は同じである。

「後戻りをしないユーザビリティの求め方」の方法を、従来の方法と比較した表を図表10-7に示す。

図表 10-7　後戻りをしないユーザビリティの求め方

フェーズ	後戻りが発生する方法	後戻りをしない方法
①企画	ユーザビリティについては何も規定しない。	画面の操作性の重要性を企画段階から確認する。
②要件定義	ユーザビリティについては何も規定しない。	画面設計の基本コンセプトを確定する。
③基本設計	画面設計は行うが、ユーザビリティのシミュレーションは実施しない。	画面設計結果に基づきシミュレーションを実施し、利用者とともに操作性の妥当性を確認する。
④詳細設計	利用者の了解がないままに画面詳細設計を行う。	利用者の了解を得た画面を基に詳細部分の設計を行う。
⑤プログラム作成（コード化)、ユニットテスト	プログラムのコード化を実施する。	プログラムのコード化を実施する。
⑥結合テスト	ベンダーSE のみで結合テストを実施する。	利用者と一緒に動く状況、レスポンス・タイム含めての確

		認を行う。
⑦総合テスト	利用者含めてテストを行い、操作性含めてのユーザビリティへの修正要望が出される。	入力データの相互干渉状況を確認する。ユーザビリティの変更は原則として行わない。
⑧稼動開始	利用開始してからもユーザビリティの変更が多発する。	利用開始してからの変更も限られたものになる。

UI 仕様書の作成

これまでに作成したタスクフロー・ダイアグラムと評価を終えたスクリーン・プロトタイプから、この情報システムで用いられるユーザ・インタフェースのデザインの詳細を「UI 仕様書」として文書化する。

この文書を基に、以降の様々な情報システム開発のための活動が行われることになる。

参考文献

[IID03a] 「ユーザビリティとは何か」、http://www.usability.gr.jp/whatis.html

[IID03b] 「ユーザビリティとは？」、http://www.usability.gr.jp/whatis/whatis001127-1.html

[ISO98b] 日本工業標準調査会審議、「人間工学－視覚表示装置を用いるオフィス作業－使用性についての手引き JIS Z 8521:1999 （ISO 9241-11:1998)」、日本規格協会、平成 11 年 3 月 20 日.

[JUA07a] 「システム・リファレンス・マニュアル 2007 年版」（社）日本情報システム・ユーザー協会、2007 年.（未刊行）

[JUA07b] 「ユーザビリティ研究会報告書 2007 年版」、（社）日本情報システム・ユーザー協会、2007 年.（未刊行）

11. 非機能要求の定義方法

非機能要求とは何か

第 3 章で、要求仕様書に書かれるべき内容として次のものをあげた[IEE98]。

1. 機能
2. 外部とのインタフェース（人間、ハードウェア、他のソフトウェアとの）
3. パフォーマンス（スピード、アベイラビリティ、レスポンス・タイム、復旧時間、など）
4. アトリビュート（移植性、正確さ（コレクトネス）、保守の容易性、安全性、など）
5. 設計上の制約

この最初の「機能」に関する要求を「機能要求」、それ以外の 4 つを「非機能要求」と呼んだ。さらに「機能（Function）とは、入力を出力に変換すること」と単純に定義し、この定義に基づいて機能要求と非機能要求の区分けを行った。

「非」は、「この次に書かれたものではない」ものを意味する。そしてこの次に書かれる単語には、一般に明確な定義が下されている。したがってこの「非」が付いた言葉は、「この明確に定義されたもの以外の全て」を意味することになる。そのためこの「非」付きの言葉は範囲が広く、領域や意味が曖昧なことが多い。

「非機能要求」も、この性格を持っている。だからもっと明確な言葉を、我々は見つけるべきだったかも知れない。しかし適切な言葉を我々は、残念ながら見つけたり作り出したりことができなかった。そのためここでは、引き続き「非機能要求」という言葉を使い続けることにする。

非機能要求の重要性

それでは、なぜこの非機能要求を我々は定義しなければならないのだろうか。それは「システムの強度」を確保するためである、というのが我々の考えである。

「システムの強度」とは、「変化／リスクに対する IT システムの強さ」を意味し、非機能要求を明確に定義することによって、この変化／リスクに対応できる情報システムを構築できることになると、我々は考える。

さらに、これまでに起きた 2 つのオープン化が、非機能要求の定義を強力に要求しているという側面もある。2 つのオープン化とは、IT 技術のオープン化と、日本経済のオープン化である。

1990 年代に起きた IT 技術のオープン化によって、メインフレームの使用を前提とした「閉じられた」アーキテクチャから、インターネットを含む「開かれた」アーキテクチャに、情報システムの根底の部分が大きく変化した。このため新しいビジネスシステム毎にインフラが用意されるという形でインフラが乱立し、BtoB、BtoC などへの対応と共にオンライン処理の対象となるトランザクション数も、そのトランザクションを入力する端末（ク

ライアント機）の数も、著しく増加するという変化が生じた。バッチ処理ですら一社の中だけではもはや完結せず、BtoB 接続やアウトソース活用などのためにオンラインと同様に分散化が進んでいる。

さらに日本経済のオープン化の結果、グローバル化、規制緩和が進み、バブル経済崩壊後の低成長の中での厳しいコスト面での制約の下、各企業は生き残りをかけて一刻を争う商品開発／サービス開発に取り組んでいる。さらにここに来て、個人情報保護への対応、J-SOX 法への対応などが、企業の情報システムに求められるという状況に至っている。

これらが前述の「システムの強度」の強化のための、情報システム化の根底にある社会と技術の両面での現時点の状況である。

ちなみに、2006 年 6 月に経済産業省から公布された「情報システムの信頼性向上に関するガイドライン[MET06b]」にも、非機能要件の重要性に関する記述がある。

非機能要求の種類

IEEE の要求仕様書に関する規格[IEE98]が定義する非機能要求の種類は、前述の通りである。しかし我々はこれをもっと詳細化し、具体化して、次のように種類分けを行った。以下ではこの種類分けに基づいて、個々の項目についての定義方法を検討する。

- 品質
- サービス・レベルに関する要件
- キャパシティに関する要件
- 安全性に関する要件
- コストに関する要件
- 保守に関する要求

非機能要求の定義の仕方

目標の定義の仕方には、「ハード・ゴール」と「ソフト・ゴール」の 2 種類がある。「ハード・ゴール」とは定量的なもの、「ソフト・ゴール」は定量的ではないものを意味する。

目標として、定量的なものが望ましいことはいうまでもない。目標が具体的な数値で示されることで、それを実現するための方法をより的確に考え、定めることができる。同時に、その目標が実現できたかどうかの判断もより容易になる。

しかし残念ながら、全て領域の全ての項目で、定量的に目標を定めることができるわけではない。この場合には、「ソフト・ゴール」で対応せざるを得ないことになる。しかしハード・ゴールで対応できる項目には、極力数値の形で目標を明確にすることが望まれる。そのため 1 つの領域の中で、項目によって「ハード・ゴール」と「ソフト・ゴール」の両方が混在することがある。

機能要求は、ビジネス上の必要性から明確にされる。非機能要求も機能要求と同様、ビジネスや IT サービスの視点から明確にされることが必要である。したがってこの要求も、ユ

ーザやシステム・オーナと、システム開発者の間で合意する必要がある。しかし単にそれだけではなく、システム稼働開始後の安定稼働に責任を持つシステム運用部門とも合意しておくことが重要である。

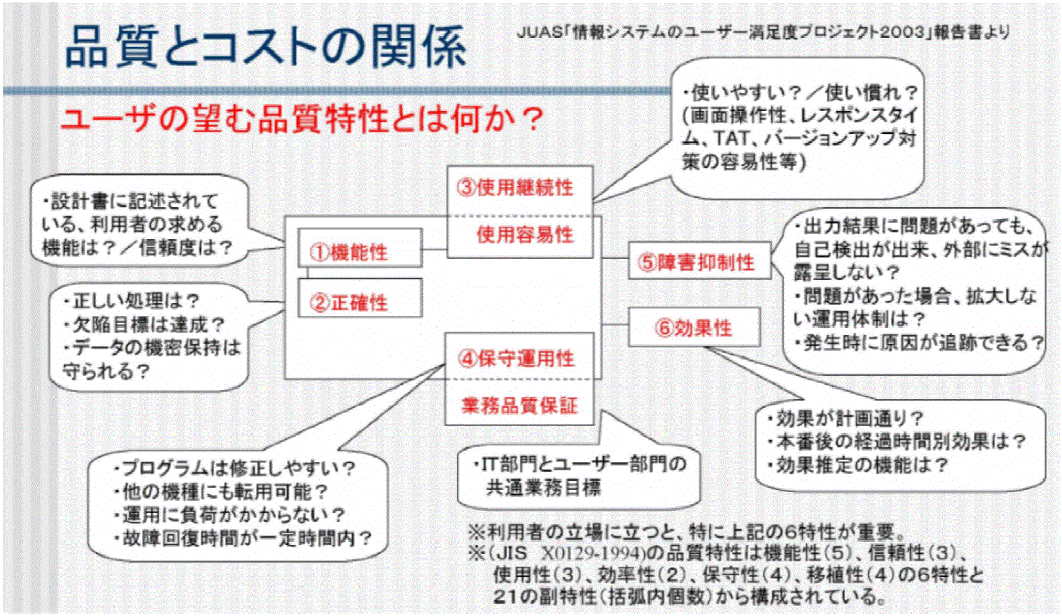
機能要求の性格から、それぞれのシステムでの共通点はさほど多くないかも知れない。しかし非機能要求では、各システムに共通する要素が多い。したがって各項目について基本的な要求を前もって定義しておき、それと異なる要求、あるいはその基本を上回る要求をする場合だけ明確に要求を記載するという方法が、この定義に当たっての現実的な対応と思われる。

以上のことからこの非機能要求については、ユーザ部門やシステム・オーナの合意を得た上で IT 部門が記述することが妥当と考える。

品質

品質要求はカットオーバ時に、1 単位のソフトウェアの大きさ、つまり KLOC（Kilo Lines of Code：プログラム 1,000 ステップ）、または FP（Function Point）あたり、いくつの欠陥が残っていても構わないとするかで表される。JUAS が提案した独自の指標として、開発費用の単位当たり（例えば 500 万円）の欠陥数というものもある。

この数値を決めることは、容易ではないかも知れない。しかし品質目標を持たないソフトウェア開発プロジェクトでは、持っているプロジェクトと比較すると欠陥率が 2 倍になったという調査結果がある[JUA06a]。より品質の高いソフトウェアを望むなら、具体的にこの数値を決め、それをクリアすることをプロジェクトの目標にすることが望ましい。ただし現状では、これを契約条件にまで取り込むことは難しいかも知れない。



図表 11-1　ユーザの望む品質特性（[JUA03]より）

さらにJUASでは2003年に、ユーザの立場からの品質特性を定義している[JUA03]。この品質特性を図表11-1に示す。本来的には、ここで定義するべき品質目標はこの図にある6項目（機能性、正確性、使用継続性、保守運用性、障害抑制性、効率性）であるべきなのかもしれない。

サービス・レベルに関する要件

サービス・レベルに関する要求とは、コンピュータに対するユーザの期待を表すものである。ユーザは一般に、大量のデータの処理や複雑な処理を、時間をかけずに瞬時にこなすことを期待している。

しかしそのようなユーザの期待に応えることは、単純にコスト面からだけ考えても容易なことではなく、場合によれば不可能なことすらあり得る。この観点から事前にユーザと充分に話し合いを行って、合意を取り付けておくことが必要である。

さらにこのサービス・レベルは、システムの構成やアプリケーションの設計の仕方で大きく異なることがある。事前に合理的な要求を明確にしておき、カットオーバ後に「こんなはずではなかった」というようなことがないようにしなければならない。

サービス・レベルに関する要求には、次のような項目が含まれる。

- サービス提供時間
- 性能
  - レスポンス・タイム
  - バッチ処理の実行時間
  - デリバリー・タイム

サービス提供時間

サービス提供時間とは、オンライン系サービスが稼働している時間帯を意味する。

ユーザの立場からすると一般に、サービス提供時間は長ければ長いほどよいことになる。しかしこの時間が長いことは運用コストの上昇を招き、単にそれだけでなく開発のワークロードや、その結果として開発のコストにも影響する場合がある。それを考慮してユーザやシステム・オーナは適正な時間を設定しなければならず、IT部門もそれを事前によく説明しておかなければならない。

場合によれば、ここで、「ベスト・エフォート」という表現が使われることがある。しかしこの表現は、極力避けるべきである。例えば「ベスト・エフォート」という表現が使われた場合、ユーザ側は「稀には障害が発生して止まるかも知れないが、自分には関係ないだろう」という期待を持つかも知れない。一方システム運用者は「安定稼働に全力を尽くすが保証はできない。サービスの提供を止めることもあり得る」と考えるとする。実際に運用が開始されてこの違いが明確になり、問題を起こすことがあり得る。

サービス提供時間設定の例として、次のような表現がある。

- オンライン稼働時間　：　8：00～20：00
- 稼働曜日　：　月曜日～日曜日（土日、祝祭日も稼働）
- 稼働日数　：　363 日（年に 2 回の保守停止がある）

性能・レスポンス・タイム

レスポンス・タイムとは、オンラインでデータなどを入力してから、結果が出力され始めるまでの時間をいう。

この時間は、アプリケーションのロジックや画面の構成などによって大きく変わるのが普通である。したがって一律にこの時間を設定することは現実的ではなく、処理毎に目標値を決めるのが妥当な方法といえる。

またこの数字は、サーバの構成、ネットワーク環境やユーザ側のシステムの構成などでも大きく変わるのが普通である。そのためこの数値を設定するに当たって、運用環境やユーザの環境などについて前提条件をシステム・オーナと取り決めておくことが必要となる。

この定義の例として、次のような表現がある。

- ログイン・メニューの遷移　：　2 秒
- 個別照会　：　2 秒
- 一覧表示（最大 20 件）　：　5 秒
- 登録処理　：　3 秒
- 複雑条件検索　：　10 秒
- 回線の前提　：　ブロードバンド。実効速度は 1Mbps 以上。
- PC スペックの前提　：　Pentium4、2GHz 以上。メモリーは 1GB。WindowsXP Professional。
- サーバ等の前提　：　本番環境。端末数も実稼働時と同じ（数千台以上のこともある）。

なおレスポンス・タイムを測定するに当たっての環境に、いくつかの種類がある。この解説を、この章末に付 11-2 として添付する。

性能・バッチ処理の実行時間

バッチ処理の実行時間はシステムの安定性、継続性に大きく影響するので、エラー発生時のリカバリーも考慮して、予め開発側と運用側で取り決めておく必要がある。またシステムによっては、他のシステムとの連携についても検討する必要がある。この他のシステムが自社だけでなく外部のシステムのケースもあり得るので、注意が必要である。

この定義の例として、次のような表現がある。

- バッチ稼働開始時間　：　24：00
- 完了限界時間　：　28：00

- 他のシステムとの依存要件　：　なし。

性能・デリバリータイム

デリバリータイムとは、ユーザが入力情報を入力し終わってから、特定の担当者の手元に適切なアウトプットが届くまでに要する作業時間全体をいう。これについては定義が漏れやすいので、注意する必要がある。

デリバリータイムを短くするとコストがかさむことがある上、発送物の品質劣化や誤送などのリスクも高まる可能性がある。これらの要素も勘案し、適切な時間／期間を決定しなければならない。

この定義の例として、次のような表現がある。

- ○○表のデリバリータイム　：　データ入力の翌々日 15 時までに、○○担当者の手元に届くこと。
- 契約者向け○○はがき　：　処理の翌日に発送できる状態にすること。

キャパシティに関する要件

昨今、システムのリソース不足が原因となったシステム障害が目立つようになった。この要因でトラブルが発生したとき、一般にリカバリーに時間がかかるという特徴があり、場合によれば利用制限をせざるを得ないことがあって、ビジネスの遂行に影響を与えることにもなりかねない。したがって、キャパシティ・プランニングの重要性を充分に認識しておかなければならない。

リソース不足を生む要因として、次のようなものを挙げることができる。

- キャパシティ要件の予測の誤り
- リソース設計の誤り
- 運用開始後の監視の不備

キャパシティに関する要求には、次のような項目が含まれる。

- データ量
- ユーザ数
- トランザクション数
- アウトプット数
- インプット数

データ量

データ量とは、システム内に格納しなければならないデータの量を指す。

これはシステム・オーナが、ビジネスの観点から事前に見積もっておく必要がある。システムの稼働開始時の量と共に、その後の推移も予測しておく必要があるが、どれくらい先まで見積もっておくかはケース・バイ・ケースで異なる。しかし少なくとも稼働開始後の

直近のピーク時（年度末など）のデータ量と、1 年後の総データ量程度は算出しておくことが望ましい。

この定義の例として、次のような表現がある。

- 契約件数　：　リリース時　1，000 千件
  12 月末（初回ピーク）　1，200 千件
  1 年後　1，500 千件

ユーザ数

ユーザ数については、ユーザの種類と、その種類毎の数について定義し、システム開発者、システム・オーナ、システム運用者の 3 者で合意を取り付けておくことが必要である。

ユーザの数はシステム上のデータベースの容量だけでなく、登録事務の体制やソフトウェアのライセンス費用などに対しても重要なファクターである。またユーザの種類は稼働開始後のトラブル発生時での影響範囲を特定する上で役立つ。

この場合も稼働開始直後の数値と共に、その後の一定期間の推移を見る必要がある。

この定義の例として、次のような表現がある。

- インターネット経由　稼働開始時　：　20 万ユーザ（不特定）
  年内ピーク時　：　30 万ユーザ（不特定）
  1 年後　：　40 万ユーザ（不特定）
- 社内ユーザ　：　常時 3 万ユーザ
- 外部（特定企業）接続　：　常時 500 ユーザ

トランザクション数

トランザクション数は、システムのキャパシティ・プランニングやリソース設計に最も大きな影響を与える重要な要件である。事前に予測し、定義することが難しいが、同時に予測を誤った場合最も大規模なトラブルに繋がりやすいファクターでもある。

また数字として何を示せばよいのかが、システムの構成や採用するアーキテクチャによって異なるという側面を持つ。例えば、次のような数字があり得る。

- システムに入力されるトランザクション数
- 単位時間あたりのページビュー（画面遷移）の数
- HTTP プロトコルのリクエスト数

また一般にトランザクションには集中する時期や時間帯があるため、年間、月間、および一日を通してのピーク時を明確にし、その時点でのトランザクション数を明示する必要がある。

この定義の例として、次のような表現がある。

- ピーク時の定義　：　年間の 20%が○月に集中
  月間の 10%が○日に集中

1 日の 20%が○時に集中

- アプリケーショントランザクション数（画面遷移数）　：　300TPS
- HTTP リクエスト数　：　1，000RPS

アウトプット数

アウトプット数とはシステムの処理を基に、印刷物やメールなどの形で最終的にアウトプットされるものの量を表す。システム内部の処理と比較して、印刷処理や発送処理は格段に時間がかかるという特徴がある。したがってこのアウトプット数は、業務設計に大きな影響を及ぼす。

この見積もりを誤ると想定した時間内に業務が終了せず、ビジネスに大きな損害を与えることがある。また発送処理を伴うアウトプット処理では、その件数が運用コストに直結することになる。

システムの設計や開発段階では、ソフトウェア技術者はデータを出力するところで検討を終えてしまうことが多い。しかし全てのアウトプットを正確、かつ適切に担当者の手元に届けるところまでを視野に入れて当たれば、システムの実現形態が変わることもある。

アウトプット数はこのように、たいへんに重要な定義項目である。

この定義の例として、次のような表現がある。

- ○○帳票のアウトプット数　印刷数　：　10，000 千枚／月
  発送数　：　1，000 千通／月
- ○○はがきのアウトプット数　印刷数　：　5，000 千枚／月
  発送数　：　5，000 千通／月

インプット数

情報システムへの入力は、その比重を急速にオンラインに移している。しかし入力原票を基に専任者が手作業で入力する一括入力は、避けられないものと考えておかなければならない。このような場合一般に、入力原票の文字種、文字のサイズなどが、業務効率に大きな影響を与える。ユーザに満足してもらえる情報システムを構築するためには、こうした点にも注意を払わなければならない。

この定義の例として、次のような表現がある。

- ○○データのインプット数　：　5，000 件／月

安全性に関する要件

ビジネスの継続は、企業が果たすべき社会的責任の中、最も重いものである。その企業のビジネスの遂行を支える情報システムの安全性の確保はこの観点から、極めて重要な要件であるといえる。

安全性を高めるには、コストがかかる。したがってリスク評価を行いながら、適切な対応

を取る必要がある。

システムの安全性には、次の 3 つの要素がある

- 機密性（Confidentiality）: 情報にアクセスすることを認可されたものだけがアクセスできることを確実にすること。
- 完全性（Integrity）: 情報及び処理方法の正確さ及び完全である状態を安全防護すること。
- 可用性（Availability）: 許可された利用者が、必要なときに情報にアクセスできることを確実にすること。

この 3 つの要素を、情報セキュリティ CIA と呼ぶ。これらは 1989 年に発行された OSI（オープン・システム・インターコネクション : 開放型システム間相互接続）に関する ISO の規格で内容が定義されており[ISO89] 、その実施は 1992 年に OECD（経済協力開発機構）が定めた情報セキュリティ・ガイドラインで勧告されている。

各企業はこれらを基に対策を抽出し、優先順位を決めて、コストも考慮しながら具体的な対応を決定する必要がある。この研究会の会員企業の中には、上記 3 要素から以下の観点で安全性要件を明確にし、対応を計っているところがある。

- セキュリティ
- 停止許容時間
- 有事対策

セキュリティ

前述の通りビジネスの継続は企業にとってたいへんに重要な要件であるが、セキュリティ対策の不備はこのビジネスの継続性を脅かす可能性がある。最近は、ウィルスやスパイウェア、クラッキングによる情報漏洩やデータ改ざんといった外部からの故意による犯罪だけでなく、開発者や運用者、あるいはユーザの過失などによって結果的に情報漏洩が起きたり、あるいはシステム停止が起きるというケースが発生している。

こうした事態を防ぐには、企業全体のセキュリティ・ポリシーを定めて、各システムをそれに準拠させることが必要である。ただし、静的なページを表示させるだけの簡易的な Web サイトと多量の個人情報を入出力するオンライン・システムに全く同じセキュリティ対策を施すことは、適切なリスク・コントロールがなされているとは言い難い。セキュリティ・ポリシーを基本の指針とし、これに則ってシステム毎にリスク評価を行いながら適切なセキュリティ要件を決めなければならない。

セキュリティ要件の中には、「パスワードの桁数や文字種」、「パスワードの有効期限」、「ユーザ・アカウントのロック条件」などのようにユーザビリティとトレードオフになるものが多い。したがってこのようなセキュリティ要件を決めるに当たっては、システム・オーナの合意を取り付けておくことが不可欠である。機密の保持が高いレベルで要求される情報システムの場合、「ワンタイム・パスワード」を採用することが必要かも知れない。

これまでログを取得する目的は、性能評価や統計処理の資料取得などが主なものだった。しかしここに来てトラブル分析などのため操作証跡を残すために組み込まれるケースが増加している。さらに最近は存在証明（整合性がとれた確実なタイムスタンプの取得)、完全性証明（改ざん防止）などのためのログ取得があり、短期間で必要な情報を抽出することが求められている。このような立場・観点でログを取得するに当たっては、膨大なりソースを必要とし、システム設計や運用コストに大きな影響を及ぼすことになる。したがって要件定義の段階で、明確に定義しておくことが必要である。

セキュリティ要件の定義例として、以下のような表現がある。

- パスワードの桁数　：　7 桁以上
- パスワードの有効期限　：　3 ヶ月間
- ユーザ・アカウントのロック条件　：　ログイン認証を 3 回連続して失敗した場合

停止許容時間

サービス時間中にシステムが停止した場合の、普及までの停止許容時間をあらかじめユーザやシステム・オーナと取り決めておくことが必要である。ユーザの立場からすると、停まらないに越したことはない。しかし高可用性を確保するためには、一般にたいへんなコストがかかる。経済合理性も考慮した、一定のレベルで合意を得ておくべきである。

停止許容時間にあわせて、稼働率も明確な数字で合意しておくことが望ましい。

停止許容時間の定義例として、以下のような表現がある。

- オンライン停止許容時間　：　年間 1 日（翌日までに復旧）
- システムの稼働率　：　99.99%

有事対策

企業にとって、「ビジネスの継続が重要」と既に述べた。特に金融機関など公共性の高い企業には、今これが強く求められている。これまでは大規模地震などを想定した対策が一般的だったが、ここに来てその想定範囲が広がり、テロやサイバーテロなどもその対象にしたディザスタ・リカバリーの機能が求められている。

ディザスタ・リカバリーの機能とは平時には使用されることがない機能であって、そのために特別の情報システムを構築することになる。これには当然コストがかかるが、そのコストもリスク対策コストの一部と認識し、対応しなければならない。

つまり企業全体のリスク・コントロールの観点から有事対策についての方針を決め、この方針に則ってシステムのリスクを評価し、個々のシステム毎にリカバリ・プランを検討し、実現することが必要である。どこまでやるかによって投資する金額が大きく異なるのが、この有事対策についての特徴の 1 つということができる。

有事対策の定義例として、以下のような表現がある。

- バックアップセンターで、n 時間以内に業務を継続する。
- バックアップデータを、週次で遠隔地保管する。

コストに関する要件

プロジェクト開始時には、開発コストやハードウェア／ソフトウェア導入のための一時コストに焦点が当たりやすい。しかしシステム稼働開始後の運用コストも、たいへん重要である。システム・オーナは「稼働開始後も、延々とコストがかかり続ける」ことを認識しなければならない。またそのために、要件定義時に運用コストについても明確な合意を取り付けることが肝要である。

コストについての定義例として、以下のような表現がある。

- 開発コスト　開発規模　：　10K ステップ、300 人月
  開発ツール　：　500 千円 × 20 ライセンス
  ・・・
- 運用コスト　：　年間 30，000 千円
- その他　：　稼働後 5 年目に、保守期限切れ対応で一時コストがかかる見込み。

保守に関する要求

非機能要求の一部として、保守についての要件を明示するべきという考えがある[SIM05]。

広い意味でこれも品質要求の 1 つといえるが、開発を委託する立場で、その情報システムの保守についてのあるべき姿を具体的に明示することが必要である。

保守についての要求例として、以下のような表現がある。

- 1 つのクラスに含まれるメソッドは、原則として 10 個以下とすること。
- モジュールの複雑度は、原則 20 以下とすること。
- モジュールの凝縮度は、原則として“手順的凝縮度”以下にはしないこと。
- 処理と管理は、明確に分離すること。
- 関数の呼び出しの深さは、原則として 5 以下に留めること。
- タスク間でアクセスし合うグローバル・データは、原則として作らないこと。
- 上記原則に違反する場合は、品質検討会議で了解を取ること。

なお、保守性についての定義、保守性を向上させるための対応方法を、付 11-3 としてこの章末に添付する。

非機能要求の適用について

機能要求とそれに関連した仕様は、そのままアプリケーションの設計に結びつく。一方の非機能要求は再構成され直して、アーキテクチャの設計に用いられることになる。

アーキテクチャとは難しい言葉だが、ここでは「アプリケーションに関わるソフトウェアを円滑に動かす仕組み」と定義しておきたい。例えば以下のような事項は全て、このアー

キテクチャの範疇に属する。仮に、今対象としている情報システムの実現形態として、クライアント／サーバ方式（Web コンピューティング）を選ぶとする（この選択自身が、すでにアーキテクチャ設計の一部である）。その上で、

- どんな種類のサーバを用意するか。
- サーバとクライアント機に、どのコンピュータを使用するか。
- サーバとクライアント機の OS に、何を使用するか。
- 全体の機能を、どのように複数のサーバ機とクライアント機で分担するか。
- その間のネットワークにどの方式の、どの帯域幅のものを使うか。
- データベースでのデータの格納方式に、何を使うか。
- データベースのバック・アップ／リカバリーは、どのように行うか。
- どの部分に市販のパッケージを使い、どの部分を手作りするか。
- ・・・・・・

アーキテクチャは、「基盤」と言い換えても良い。これらの上の問題に的確に答えを出すために、アーキテクチャを設計する人（アーキテクトと呼ばれる）は非機能要求などを基に、「基盤要件」と呼ぶ次の要件を明確にする。

- 可用性
- 性能／スケーラビリティ／キャパシティ
- 拡張性
- 管理性
- セキュリティ
- 前提・制約条件

前提・制約条件には、さらに以下のような事項が含まれる。

- 使用を前提とする市販の製品
- 既存システムとの接続
- アプリケーションの仕様
- 運用要件
- 保守要件
- 機器等の設置環境
- 法規制
- ・・・・・

良いアーキテクチャはアプリケーションのソフトウェアにとって、良い稼働環境などを提供するものである。単にアプリケーションの稼働環境だけではなく、情報システムとしての拡張性、ユーザビリティ、セキュリティ、障害の起きにくさと復旧の容易さ、それらを総合した運用の容易さなど、非常に広範囲に影響を及ぼすことになる。逆説的な言い方になるが、あるいはその存在を意識させないアーキテクチャが、一番優れたアーキテクチャということになるのかも知らない。

このように非常に大切なアーキテクチャ設計のベースになるものであるから、非機能要求も機能要求と同様、業務の特質をふまえ、しっかりと定義しなければならない。

開発リスク軽減の方法

ソフトウェアの開発を外部に委託する場合、その開発にはリスクを伴う。

開発を受託するソフトウェア会社の立場からすると、そのリスク分は開発受託の金額の一部として含まれており、リスクが高いほど受託金額が高くなるということになる。したがって発注者側と受託者側で共同してこのリスクを減少させることができれば、双方にとってメリットがあることになる。

当研究会のメンバー企業の 1 つに、あるソフトウェア会社がある。その会社では開発受託に伴うリスクを「開発規模が増大（あるいは減少）するリスク」と「生産性が低下（あるいは向上）するリスク」に分けてそれぞれをまず評価し、それらから最終的な開発のリスクを求めるという方法をとっている[JUA05]。

その会社が現在リスク評価に使用している表を、この章末に付 11-1 として添付する。ここでいう「規模環境変数」とは規模が増加（あるいは減少）する要因を抽出し、最終的に規模の増加（減少）に関わるリスクを求めるもの、「生産性環境変数」とは生産性を低下（あるいは向上）させる要因を抽出し、生産性の低下（あるいは向上）に関わるリスクを求めるものである。この表に記載する項目は、いずれも機能要求、あるいは非機能要求として要求仕様書で提示したもの、あるいはそれから導き出すことができるものである。

ソフトウェア開発を委託する側自身がこれらを評価してリスクを確定することは、現実は困難なことかもしれない。したがってここに添付する付 11-1 の諸表は、リスク分析の考え方を提示した 1 つのサンプルとしてとらえておいて欲しい。ただ、前述の通りリスクを軽減させることは開発の委託者側にも受託者側にもメリットがあることであるから、ソフトウェアの開発を委託する場合には受託するソフトウェア会社にこのような形でのリスク評価を依頼し、ここに添付したような資料を用いて双方でリスクの要因を分析し、委託者としてここで軽減できるリスクは積極的に軽減することを是非お勧めしたい。

（注）

この章は、当初は「品質要求の定義方法」という標題を予定していた。しかし議論の過程で、標題を「非機能要求の定義方法」に変更した。

参考文献

[IEE98] IEEE-SA Standards Board, “IEEE Recommended Practice for Software Requirements Specifications IEEE Std 830-1998”, The Institute of Electrical and Electronics Engineers, Inc., 1998.

[ISO01] 日本工業標準調査会審議、「ソフトウェア製品の品質－第 1 部：品質モデル JIS

X 0129-1：2003 （ISO／IEC 9126：2001）」、日本規格協会、平成 15 年 2 月 20 日.

[JUA03] 「情報システムのユーザ満足度プロジェクト報告書」、（社）日本情報システム・ユーザー協会、2003 年.

[JUA05] 「システム・リファレンス・マニュアル（SRM）」、（社）日本情報システム・ユーザー協会、2005 年 9 月.

[JUA06a] 「ユーザー企業ソフトウェアメトリクス調査 2006」、（社）日本情報システム・ユーザー協会、2006 年.

[MET06b] 「情報システムの信頼性向上に関するガイドライン」は、次の URL からダウンロードできる。

http://www.meti.go.jp/press/20060615002/guideline.pdf

[SIM05] 清水吉男著、「[入門+実践]要求を仕様化する技術表現する技術～仕様が書けていますか」、（株）技術評論社、平成 17 年.

付 11-1　規模環境変数と生産性環境変数

規模環境変数

生産物量見積方式　　$V_i = V^B_i \times (1 + \Sigma \alpha_{ij})$　　α_{ij}：生産物量環境変数

品質特性	副品質特性	評価の観点 内容	外責評価基準（見積プレゼン用） 影響度（%） 要件定義	設計	製作	テスト		内責評価基準（参考） 影響度 設計	製作	テスト	
機能性	合目的性(V00)	下記要因による機能要求の増加を該当事象数（a～d）により評価する	－	－	－	－	（注）各事象の影響度により、補正する				
		a. 利用者の所在の広がり(ﾊﾞｯｸｵﾌｨｽ→ﾌﾛﾝﾄｵﾌｨｽ→一般ﾕｰｻﾞ)および利用時間の増加	0	0	0	0	該当する事象はない				
		b. ｽﾃｰｸﾌｫﾙﾀﾞｰの増加に伴う規模の増加（保有ｼｽﾃﾑ規模で評価）	10	10	10	10	該当事象数　１				
		c. ｺﾝﾃｨｼﾞｪﾝｼｰに対応する機能要求の付加	30	30	30	30	該当事象数　2～3				
		d. 不正な移行元ﾃﾞｰﾀに対する配慮（例外処理など）	50	50	50	50	該当事象数　４以上				
	正確性(V01)	・実現されている機能が正常に動作することに対する要求	－	－	－	－	－				
		・前工程の機能が当該工程で正しく実現されていること、生産物間および機能間に	－	－	0	0	標準的なテスト項目により確認・検証可能				
		矛盾がないことを確認・検証する度合を評価する	－	－	10	20	標準に対し１．２倍のテスト項目を要す				
		・具体的にはテスト密度で評価する	－	－	15	30	標準に対し１．５倍のテスト項目を要す				
		標準テスト密度はテスト項目ガイド（別途）で定める	－	－	20	50	標準に対し２．０倍以上のテスト項目を要す				
	相互運用性(V02)	・ソフトウェアが他ソフトウェアや他システムとデータやコマンド等をやりとり	－	－	－	－	－				
		できる度合いに対する要求であり、コード変換、フォーマット変換の設計／	0	0	0	0	変換数　　０				
		実装が必要になる	2	10	10	10	変換数　　～５				
		・コード変換、フォーマット変換数で評価する	3	15	15	15	変換数　　５～１０				
			5	20	20	20	変換数　　１１以上				
	標準適合性	・開発量の増減事例として環境基準に対応する法規制および条例がシステムに	－	－	－	－	－				
		適用する要求で、環境配慮設計および実装を必要とする付加機能を検討する	0	0	0	0	法規制および条例の数　０件				
			2	2	2	2	法規制および条例の数　１件				
			4	4	4	4	法規制および条例の数　２件				
			5	5	5	5	法規制および条例の数　３件以上				
	セキュリティー(V03)	・ソフトウェアが情報の漏洩・紛失・外部からの不正使用やシステム資源の	－	－	－	－	－				
		破壊等を防止あるいは検出できることに対する要求で、アクセス管理機能や	0	0	0	0	実現機能数　０ ～ １				
		記録管理機能の設計／実装が必要となる	10	10	10	10	実現機能数　２ ～ ３				
		・対応が必要なセキュリティ機能の数で評価する	15	15	15	15	実現機能数　４ ～ ５				
		※機能要件に定義されている部分は除く。詳細は、補足資料参照。	20	20	20	20	実現機能数　６以上				
信頼性	成熟性(V04)	・ソフトウェアシステムに障害が起きた場合に、他システムコンポーネントへの	－	－	－	－	－				
		影響を遮断できることや、２次障害の影響を少なくできることに対する要求	0	0	0	0	実現機能数　０				
		・対応が必要な故障低減機能の数で評価する	2	3	3	3	実現機能数　１				
			3	6	6	6	実現機能数　２				
			5	10	10	10	実現機能数　３以上				
	障害許容性(V05)	・障害が発生しても、それをダウンとして顕在化させないことに対する要求で	－	－	－	－	－				
		異常が顕在化しないうちに再試行する機能やチェック強化によるダウン発生の	0	0	0	0	実現機能数　０				
		回避機能の設計／実装が必要となる	2	3	3	3	実現機能数　１				
		・対応が必要な異常検知機能の数で評価する	3	6	6	6	実現機能数　２				
			5	10	10	10	実現機能数　３以上				

	回復性(V06)	・ソフトウェアシステムがダウンしてから再稼動し、処理再開するまでの時間が短いことに対する要求 ・対応が必要な再開処理機能の数で評価する	－	－	－	－	－				
			0	0	0	0	実現機能数　0				
			2	3	3	3	実現機能数　1				
			3	6	6	6	実現機能数　2				
			5	10	10	10	実現機能数　3以上				
使用性	理解性(V07)	・ソフトウェアの機能、働きが分かりやすいことに対する要求 ・作成するプレゼンテーションツールの数で評価する （留意）理解性を向上させる為の操作マニュアル作成アクティビティのみ対象 （影響度：0～100%）	－	－	－	－	－				
			0	0	－	－	作成対象数　0				
			1	3	－	－	作成対象数　1				
			2	6	－	－	作成対象数　2				
			3	10	－	－	作成対象数　3以上				
	習得性(V08)	・ソフトウェアの使い方が学びやすいことに対する要求 ・作成するマニュアルの数で評価する （留意）習得性を向上させる為の操作マニュアル作成アクティビティのみ対象 （影響度：0～100%）	－	－	－	－	－				
			0	0	－	－	作成対象数　0				
			1	3	－	－	作成対象数　1				
			2	6	－	－	作成対象数　2				
			3	10	－	－	作成対象数　3以上				
	操作性(V09)	・使用者からソフトウェアへの働きかけが簡単にでき、かつ分かりやすく心理的／肉体的に疲れにくくなっていることに対する要求 （ソフトウェアの運用の容易さ、インストールの容易さを含む） ・対応が必要な操作性向上機能の数で評価する	－	－	－	－	－				
			0	0	0	0	実現機能数　0 ～ 1				
			3	10	10	10	実現機能数　2 ～ 3				
			6	15	15	15	実現機能数　4 ～ 5				
			10	20	20	20	実現機能数　6以上				
保守性	解析性(V10)	・故障または運用上の不都合が発見された場合、どの程度労力をかけることなく原因の解析ができるかに対する要求 ・対応が必要な解析機能の数で評価する	－	－	－	－	－				
			0	0	0	0	実現機能数　0 ～ 1				
			2	3	3	3	実現機能数　2 ～ 3				
			3	6	6	6	実現機能数　4 ～ 5				
			5	10	10	10	実現機能数　6以上				
	変更作業性(V11)	・ソフトウェアの変更実施が容易であることに対する要求 ソフトウェアの目的、働き、特徴等の外部から見た理解の容易さと、構造やアルゴリズム等の内部の実現方法の理解の容易さを含む ・作成する保守用ドキュメントの数で評価する （留意）操作マニュアル作成アクティビティのみ対象（影響度：0～200%）	－	－	－	－	－				
			0	0	－	－	作成対象数　0				
			1	3	－	－	作成対象数　1				
			2	6	－	－	作成対象数　2				
			3	10	－	－	作成対象数　3以上				
	試験性(V12)	・ソフトウェアのテストや性能、効率等の特性の評価が容易であることに対する要求（準備、実行のための労力や、ソフトウェアの実行状況の測定を含む） ・対応が必要な試験機能の数で評価する	－	－	－	－	－				
			0	0	0	0	作成対象数　0				
			1	3	3	3	作成対象数　1				
			2	6	6	6	作成対象数　2				
			3	10	10	10	作成対象数　3以上				

（注１）移行、教育、保守および運用作業は、規模環境変数の適用対象から除いている。
　　但し、ソフトウェア開発において、保守、移行、運用等で効率性が向上する支援ドキュメントの作成は、設計工程にオプションとして含めている。
（注２）内責評価基準はジャステック内部事情で発生する環境変数ゆえ、今回は除いている。
（注３）評価基準の影響度は実運用上、工程毎のアクティビティ（作業）単位に影響度を設定している。今回、工程は要件定義、設計、製作、テストと簡略化し、アクティビティも省略している。

（補足資料）機能性・セキュリティに関するセキュリティ機能

IS015408で定めるセキュリティ機能要件に基づいて、ソフトウェアの機能要件として識別する要件と
ソフトウェアの非機能要件として識別する要件とを下表の通り整理する。
規模の環境変数として評価する「セキュリティ機能」とは、ソフトウェアの非機能要件を指す。

No.	セキュリティ機能要件	S/Wの機能要件	S/Wの非機能要件
1	セキュリティ監査 セキュリティ事象に関連した情報の認識、記録、保存分析に関する要件	○	
	① セキュリティ監査の自動警告応答機能		
	② セキュリティ監査データ（ログ情報）生成機能		
	③ セキュリティ監査データ自動分析機能		
	④ セキュリティ監査レビュー支援機能		
	⑤ セキュリティ監査対象イベントの選択機能		
	⑥ セキュリティ監査証跡を安全に生成・維持する機能		
2	通信における否認防止機能 データ通信への参加者の識別を保証する否認防止に関する要件		○
	① 発信者の送信否認防止		
	② 受信者の受信否認防止		
3	暗号サポート 暗号鍵の生成／配布／失効の管理、データの暗号化／復号、デジタル署名 の生成／検証などの暗号操作に関する要件		○
	① 暗号鍵管理機能		
	② 暗号操作機能		
4	利用者データの保護		○
	① 利用者データ転送時のセキュリティ属性保護機能		
	② 利用者データのインポート時のセキュリティ属性保護機能		
	③ 蓄積データ完全性保護機能		
	④ 利用者データ転送時の機密性保護機能		
	⑤ 利用者データ転送時の完全性保護機能		
5	識別と認証 利用者のアイデンティティを確立し検証する要件	○	
6	セキュリティ管理機能 セキュリティ属性やセキュリティ機能に関するデータ（例：認証データ、 セキュリティ方針ＤＢ）などの管理に関する要件	○	
7	プライバシー保護機能 他者のアイデンティティの発見と悪用（探り出し防止）に関する要件	○	
	① 匿名利用機能		
	② ペンネームによる利用機能		
	③ 利用形態のユーザアイデンティティからの独立性		
	④ 利用時他ユーザからの非観察性		
8	資源利用 資源の耐障害性、優先制御、資源割当てに関する要件		○
	① 障害耐久性		
	② 資源割り当ての優先制御		
	③ 資源割り当て管理		
9	情報資源へのアクセス制御 ユーザセッション確立の制御の関する要件 ① 同一利用者による服す同時セッション数の制限 ② インタラクティブセッションのロック・アンロック機能 （接続時間制限機能を含む） ③ 情報資産へのアクセス履歴管理機能		○

生産性環境変数

生産性見積り方式　$P_i = P^{B}{}_i \times (1+\Sigma \beta_{ii})$　β_{ii}:生産性環境変数

生産性特性	副生産性特性	評価の観点 内容	特記、特例事項	外資評価基準（見積プレゼン用） 影響度（%） 要件定義	設計	製作	テスト		内資評価基準（参考） 影響度 設計	製作	テスト	
業務特性	業務ナレッジ(P01)	・顧客／当社の開発対象業務に対する業務ナレッジが生産性に及ぼす影響を、それぞれ（外資／内資）個別に評価する ・外資評価はユーザー部門を含む顧客プロジェクト組織全体の能力として評価する ・内資評価は当該プロジェクトへのアサインベースで評価する	→世の中に無い業務モデル等の創造、システム化の要素も含む。 → プロジェクト全体としての方針・仕様意思決定能力（旧組織複合度の要素を取り込む） → 実／予定アサインメンバーの業務ナレッジで評価	-10	-10	－	-10	中核メンバー全員が当該業務経験多数				
				-5	-5	－	-5	中核メンバーの一部の当該業務経験少				
				0	0	－	0	中核メンバーの一部に未経験者あり				
				40	8	－	8	中核メンバーに当該業務経験者不在				
				50	10	－	10	メンバー全員が当該業務経験なし				
ハードウェア特性	安定度/信頼度/使用実績(P02)	・システムもしくは製品となるハードウェアの安定度・信頼度を外資として評価し、当社使用実績を内資として評価する （注）「使用実績がある」ということは、ハードウェアの特性を把握し、その対策は既知であるが故に、生産性が高いということ。	→ 実／予定アサインメンバーの使用実績で評価	-5	-5	－	-5	当該HWでの効果的開発手法を保有				
				-3	-3	－	-3	当該HWでの開発実績多数（安定度高）				
				0	0	－	0	当該HWでの開発実績が当社基準での平均				
				3	3	－	3	顧客に使用実績のないHW				
				5	5	－	5	世間で使用実績のないHW				
ソフトウェア特性	安定度/信頼度/使用実績(P03)	・システムもしくは製品となる他社作成ソフトウェアもしくはCotsの安定度・信頼度を外資として評価し、当社使用実績を内資として評価する （注）「使用実績がある」ということは、ソフトウェアもしくはCotsの特性を把握し、その対策は既知であるが故に、生産性が高いということ。	→ 実／予定アサインメンバーの使用実績で評価 →要件定義の「－」は、ERPを利用する場合を想定していないことに注意すること。	-5	-5	－	-5	当該ＳＷでの効果的開発手法を保有				
				-3	-3	－	-3	当該ＳＷでの開発実績多数（安定度高）				
				0	0	－	0	当該ＳＷでの開発実績が当社基準で平均				
				3	3	－	3	聞いたことはあるが特性がわからないＳＷ				
				5	5	－	5	聞いたこともないＳＷ				
コミュニケーション特性	顧客窓口特性(P04)	・問い合わせに対するレスポンス、約束期限の遵守度合い、方針・仕様に関する決定が覆る度合により評価する	→組織としての特性を評価するものであり、窓口役の個人を評価するものではない。（窓口が複数あると受ける側は大変で、取りまとめ役がいないと混乱し、生産性は低下する。）	-10	-10	－	-7	期限どおりに方針決定して頂ける				
				-5	-5	－	-4	ほぼ期限どおりに方針決定して頂ける				
				0	0	－	0	ほぼ期限どおりに方針決定して頂けるが、多少覆ることがある				
				10	5	－	4	期限の遅延、決定事項が覆る事が多い				
				20	10	－	7	概ね期限は遅延し、決定事項が頻繁に覆る				
	工期の厳しさ(P05)	・システム規模に対する工期の厳しさが生産性におよぼす影響を、外資／内資別に評価する。 ・２者間契約に適用するので、顧客が認識するプロジェクト全体ではなく、ベンダの担当分に焦点を当てて評価する。	→「工期の厳しさ」を以下の基準式を標準として評価する 基準工期（月）＝２．７×（人月）$^{1/3}$ （ＪＵＡＳソフトウェアメトリックス調査２００５年版参考）	—	—	—	—	—				
				0	0	0	0	基準工期に対し▲5%未満				
				3	3	1	2	基準工期に対し▲5%～▲10%以内				
				6	6	3	4	基準工期に対し▲10%～▲20%未満				
				10	10	5	7	基準工期に対し▲20%～▲30%以内				
	コミュニケーション基盤(P06)	・コミュニケーションの速度および精度を低下させる物理的な要因を評価する ・評価観点は以下のとおり a. 開発拠点（分散度）　b. 資料・開発生産物の情報共有の基盤 c. グループウエア・電子メールによる情報共有の基盤	→ 内資は開発拠点（分散度）のみ → 基盤があっても秩序がない場合は評価しない「秩序がない」とは、例えばメーリングリストによる大量情報の垂れ流しのように、決定したことがきちっと伝達される仕組みが欠如していることを指す。	-10	-5	-3	-3	利用制約のない基盤が確保され、適時適切に意思確認できる。				
				-6	-3	-1	-1	利用制約のない基盤が確保されている				
				0	0	0	0	基盤は確保されているが利用制約がある				
				6	3	1	1	指示が徹底されたことを確認する仕組みが欠けている				
				10	5	3	3	拠点が分散し、且つ、必要な基盤が確保されていない				
	レビュー体制(P07)	・共同レビューの目的・目標に照らして、レビュー体制およびレビュー方法の適切性（効率および欠陥除去率などの品質向上への寄与率）を評価する ・体制：過剰なレビュー参加者要求はコスト増加に対して効果が少ない ・方法：多段階レビュー（担当者間→上司→その上司など）によるレビュー効率低下（なかなか収斂せず、結論が覆ることが多い）	→体制：開発者全員によるレビュー実施など対窓口レビュー以外のレビュー参加要請など	-5	-5	-3	-5	レビュー観点の整理および事前査読によりレビュー時間を短縮				
				-3	-3	-1	-3	事前に査読してレビュー時間を短縮				
				0	0	0	0	適切な体制、方法で共同レビューを実施				
				3	3	1	3	多段階もしくは関連範囲外へのレビュー参加要求				
				5	5	3	5	多段階かつ関連範囲外へのレビュー参加要求				
開発環境特性	開発手法／開発環境(P08)	・開発手法・開発環境（ソフト・ハード・ツール）について評価する ・評価観点は以下のとおり（「外資」にはベンダの習熟度の評価は含まない） a. 機能充足度　b. 信頼性（安定性・使用実績） c. 操作・運用マニュアル具備状態　e. 占有率(H/W等の占有利用可能の程度) d. 同時開発(ﾂｰﾙ、ｲﾝﾌﾗなどとｱﾌﾟﾘの同時開発)	→ 既存テスト環境流用度合はツールとして評価する（標準試験項目セット有無も） → 要件定義工程は要件定義手法・ツールなどを評価する。	-3	-3	-5	-5	生産性向上の工夫が盛込まれた手法／環境				
				-1	-1	-3	-3	一般的手法／環境であり安定度高				
				0	0	0	0	一般的手法／環境であるが評価観点での不適あり				
				1	1	3	3	聞いたことはあるが特性がわからない手法／環境				
				3	3	5	5	聞いたこともない手法／環境				

	テスト手順書水準(P09)	・テスト手順の具体化度（操作手順&入出力の具体化）を評価する		−	−	−	−	−					
				−	−	−3	−3	テスト項目、確認ポイントに限定して記載					
				−	−	0	0	キー項目実現値の記載を要求					
				−	−	2	2	入出力データ実現値すべての記載を要求					
				−	−	3	3	入出力データ実現値および手順の記載を要求					
工程入力情報特性	業務関連資料(P10) 他システム関連資料(P11) 規約・標準化関連資料(P12)	・既存資料についてはその信用度（正確性、最新度）を評価する 並行作成するものならば、上記に加え期日厳守率をあわせて評価する ・他システム関連資料は当社がＩ／Ｆ設計およびテスト設計を実施する上で必要十分かつ正確な情報であるかを評価する ・規約・標準化資料は複数存在していたり不整合がないか評価する	評価規準は以下の事項に着目して設定している ・資料の整理のされ方 ・資料の読み易さ ・情報の探し易さ 要件定義の影響度は、以下の３要素を単純に加算した結果です。 ・業務関連資料：−10〜+10 ・他ｼｽﾃﾑ関連資料：−10〜+10 ・規約・標準関連資料：−7〜+7	−27	−7	−3	−3	検索しやすさを考慮し整備されている					
				−16	−4	−1	−1	必要資料はすべて整合性をもって整備されている					
				0	0	0	0	必要資料の一部に不整合がある					
				16	4	1	1	必要資料は整備されているが不整合が散見する					
				27	7	3	3	資料の不足があり他資料からの補完が必要					
生産性特性	副生産性特性	評価の観点		外資評価基準（見積プレゼン用）					内資評価基準（参考）				
		内容	特記、特例事項	要件定義	設計	製作	テスト		設計	製作	テスト		
顧客の協力特性	役割分担特性(P13)	・顧客がベンダに協力する度合および顧客とベンダとの役割分担の明確性を評価する	・ベンダが直接エンドユーザと交渉して開発する場合	−	−	−	−						
			・エンドユーザとベンダの間に、情報システム部が入り、エンドユーザと交渉した上で、ベンダに伝える場合とでは、ベンダ側の生産性は大分異なる。	−	−	−	−						
				−10	0	−	0	顧客側に情報ｼｽﾃﾑ担当窓口があり、ﾍﾞﾝﾀﾞは情報ｼｽﾃﾑ担当窓口と交渉する					
				0	5	−	5	ｴﾝﾄﾞﾕｰｻﾞの代表窓口とﾍﾞﾝﾀﾞとが直接交渉する					
				20	10	−	10	ｴﾝﾄﾞﾕｰｻﾞの代表窓口はなくﾍﾞﾝﾀﾞが直接交渉する					
品質特性	副品質特性	評価の観点		外資評価基準（見積プレゼン用）					内資評価基準（参考）				
		内容	特記、特例事項	要件定義	設計	製作	テスト		設計	製作	テスト		
機能性	合目的性(P18)	・ソフトウェアがユーザーニーズを満足させるために必要十分な機能を備えていることに対する要求（要件定義フェーズを対象とする） a. 新規性(世の中にないものの考案)　b. 方針の明確性 c. ステークフォルダの納得性(ＲＯＩの明示度合) d. ステークフォルダの多様性(ﾊﾞｯｸｵﾌｨｽ, ﾌﾛﾝﾄｵﾌｨｽ, 一般ﾕｰｻﾞ/外部企業など)		−				−					
				0				既存業務で、システム化の目標が定まっている					
				20				システム化の方針が曖昧である					
				50				ｽﾃｰｸﾌｫﾙﾀﾞが多様かつｼｽﾃﾑ化の方針が曖昧である					
				100				上記に加えて、世の中にないものの考案が多い					
	（要求仕様の網羅性）	・ソフトウェアがユーザーニーズを満足させるために必要十分な機能を備えていることに対する要求（設計〜テストフェーズを対象とする） ・ユーザーニーズの具体性、網羅性、ステークフォルダの同意度合を評価し、要求仕様書にユーザーニーズが具体的に記載されていない場合は、以降の工程でニーズの具体化・検証作業が必要になる	→要求仕様書の記述水準は標準ドキュメント記述文字数に対する実要求仕様書記述文字数で評価できる網羅性、ステークフォルダの同意度合を加味して評価 → 未解決懸案項目数により評価レベルを調整する → ユーザーヒアリング実施、プロトタイピング実施など	−	−	−	−	−					
					0	−	0	必要項目・水準が網羅された要求仕様書が存在					
					5	−	3	必要項目・水準のどちらかに多少、不備有り					
					15	−	6	必要項目・水準の両方に多少、不備有り					
					30	−	10	要求仕様書が不存在(代替:口頭説明,代替資料)					
	正確性(P19)	・実現されている機能が正常に動作することに対する要求 ・前工程の機能が当該工程で正しく実現されていること、生産物間および機能間に矛盾がないことを確認・検証する度合を評価する ・具体的にはレビュー充実度で評価するが、上位工程においては生産物作成過程における確認・検証作業も考慮する	→ レビュー工数の標準は各工程の全体工数の10%程度とする	−	−	−	−	−					
				0	0	0	0	標準的なレビューにより確認・検証可能					
				2	2	1	2	標準に対し１．５倍のレビュー工数を要す					
				3	3	2	3	標準に対し２．０倍のレビュー工数を要す					
				5	5	3	5	標準に対し３．０倍以上のレビュー工数を要す					
	相互運用性(P20)	・ソフトウェアが他ソフトウェアや他システムと容易に接続・運用できることに対する要求で、データの共通化や通信手段・インターフェイスの共通化検討が必要になり、社外と接続するか否かも影響する ・上記を考慮して設計すべきインタフェース先の数で評価する 設計の難易度とインターフェーステストの複雑さに影響する	→インタフェース数の増加は規模の環境変数で評価 →インタフェース数は共通化により抑えることができるが、接続先が多いと調整負荷が増加する →留意事項：保有システム規模が大きい程、インタフェース先の数も増加する傾向にある	−10	−5	−	−5	社内のｲﾝﾀﾌｪｰｽのみでｲﾝﾀﾌｪｰｽ先が1件/100KS以内					
				5	−3	−	−3	社内のｲﾝﾀﾌｪｰｽのみでｲﾝﾀﾌｪｰｽ先が3件/100KS以内					
				0	0	−	0	社外とのｲﾝﾀﾌｪｰｽを含みｲﾝﾀﾌｪｰｽ先が3件/100Ks以内					
				5	3	−	3	社外とのｲﾝﾀﾌｪｰｽを含みｲﾝﾀﾌｪｰｽ先が7件/100Ks以内					
				10	5	−	5	社外とのｲﾝﾀﾌｪｰｽがありｲﾝﾀﾌｪｰｽ先が8件/100KS以上					
	標準適合性(P21)	・実現されている機能が公的規則・規格・基準に一致し、正常に動作することに対する要求で、従うべき規則・基準が多いほど整合性の確認・検証作業が必要になる ・金融監督庁、会計監査法人等が行う「外部監査基準」に基づく要求も本副品質特性で評価する	→他ｼｽﾃﾑ、他ﾌﾟﾛｼﾞｪｸﾄとの方針整合性の保持などの全体最適性も含む。 →ｸﾞﾛｰﾊﾞﾘｾﾞｰｼｮﾝ(多言語対応、多通貨対応、多国制度対応)を含む。 → 地球環境基準への適合数により評価	−10	−5	−3	−3	整合をとる規則・規約が0件					
				−5	0	0	0	整合をとる規則・規約が1件					
				0	3	1	1	整合をとる規則・規約が2件					
				5	5	3	3	整合をとる規則・規約が3件以上					
				10	10	5	5	全体最適性の再構築が必要					

品質特性	副品質特性	評価の観点 内容	評価の観点 特記、特例事項	外責評価基準（見積プレゼン用） 要件定義	設計	製作	テスト		内責評価基準（参考） 設計	製作	テスト	
効率性	実行効率性(P22)	・定められた条件下で所定の処理を実行する早さに対する要求		－	－	－	－	－				
		・具体的には、処理を要求してから結果が得られるまでの速さや単位時間に遂行される仕事の量を示す	→ 実現方式の検討要否および事例有無により評価	0	0	0	0	一般的要求水準で既知の事例にて実現が可能				
				2	3	2	3	既知の最適事例の１．２倍の速さを要求				
		・要求が高ければ設計工程における実現可能性検証やテスト工程における障害解析・対応作業の難易度が高まる	→ 机上検証、ベンチマークテスト、パイロット開発など	3	6	3	6	既知の最適事例の１．５倍の速さを要求				
				5	10	5	10	既知の最適事例の２．０倍以上の速さを要求				
	資源効率性(P23)	・定められた条件下で所定の処理を実行する際、資源を有効に使用することに対する要求		－	－	－	－	－				
				0	0	0	0	一般的要求水準で既知の事例にて実現が可能				
		・具体的には、プログラムの実行に必要な資源の量、ハードウェア資源の使用の有効性に対する要求	→ ＣＰＵ使用率、主記憶使用量、ファイル使用量、ネットワーク使用量に対する要求	2	3	2	3	既知の最適事例の１．２倍の有効性を要求				
				3	6	3	6	既知の最適事例の１．５倍の有効性を要求				
			→ 実現方式の検討要否および事例有無により評価	5	10	5	10	既知の最適事例の２．０倍以上の有効性を要求				
保守性	解析性(P24)	・故障または運用上の不都合が発見された場合、どの程度労力をかけることなく原因の解析ができるかに対する要求	→ ソースコードのコード化規約に着目して、コメントに対する要求水準により評価する	－	－	－	－	－				
				－	－	0	－	機能概要を記述するヘッダコメントを付加				
		・解析用機能は規模環境変数として定義し、生産性環境変数としてはソースの解析性向上のための作業（リバースエンジニアリング用情報付加等）を対象	→ 旧コメント率、ＪａｖａＤｏｃ用コメントなどに対応し名称を変更	－	－	2	－	上記に加え、50行程度毎にﾌﾞﾛｯｸｺﾒﾝﾄを付加				
				－	－	3	－	上記に加え、20行程度毎にﾌﾞﾛｯｸｺﾒﾝﾄを付加				
				－	－	5	－	ﾘﾊﾞｰｽﾂｰﾙで仕様を作成できる程度のｺﾒﾝﾄを付加（ｺｰﾄﾞを読めない人でも処理内容を理解できる）				
	安定性(P25)	・ソフトウェアに変更を施した場合、システム全体の品質がレベルダウンしないことに対する要求 拡張性（10年使えるシステム等）も本副品質特性で評価する	→将来変更が発生する確率が高いと想定される場合 設計上将来の変更容易性（変更箇所の局所化を含む）考慮しなければならない	－	-3	-3	－	システムのライフサイクル目標が２年未満				
				0	0	0	－	システムのライフサイクル目標が５年未満				
				2	2	2	－	システムのライフサイクル目標が７年未満				
		・当該要求を実現するために、上流工程においてデータ正規化やオブジェクト指向などの設計手法を取り入れることが必要となる場合もある		3	4	3	－	システムのライフサイクル目標が７年～１０年				
				5	7	5	－	システムのライフサイクル目標は１０年超				
移植性	環境適用性(P26)	・多様なハード、ソフト、運用環境に適用させる要求の水準	→ 要求の数（種類）で評価する	－	－	－	－	－				
				0	0	0	0	移植要求なし				
				2	2	1	2	移植要求が１種類				
				4	4	2	3	移植要求が２～３種類				
				7	7	3	5	移植要求が４種類以上				
	移植作業性(P27)	・環境を移す際に、必要な労力を低減させる要求の水準	→ 要求の数（種類）で評価する → インフラ変更に対する対応容易性など	－	－	－	－	－				
				0	0	0	0	移植要求なし				
				2	2	1	2	移植要求が１種類				
				4	4	2	3	移植要求が２～３種類				
				7	7	3	5	移植要求が４種類以上				
	規格準拠性(P28)	・移植性に関する国際／国内規格または規約を遵守する要求の水準	→ 要求の数（種類）で評価する	－	－	－	－	－				
				0	0	0	0	移植要求なし				
				2	2	1	2	移植要求が１種類				
				4	4	2	3	移植要求が２～３種類				
				7	7	3	5	移植要求が４種類以上				
	置換性(P29)	・使用環境／条件を変更せずに他のソフトウェア製品と置き換えて使用可能とする要求の水準	→ 要求の数（種類）で評価する	－	－	－	－	－				
				0	0	0	0	移植要求なし				
				2	2	1	2	移植要求が１種類				
				4	4	2	3	移植要求が２～３種類				
				7	7	3	5	移植要求が４種類以上				

（注１）移行、教育、保守、運用作業は、生産性環境変数の適用対象から除いている。
（注２）内責評価基準はベンダー内部事情で発生する環境変数ゆえ、今回は除いている。
（注３）評価基準の影響度は実運用上、工程毎のアクティビティ（作業）単位に影響度を設定している。今回、工程は要件定義、設計、製作、テストと簡略化し、アクティビティも省略している。

生産性環境変数（改造型固有）

生産性特性	副生産性特性	評価の観点 内容	特記、特例事項	外資評価基準（見積プレゼン用） 影響度（%） 要件定義	設計	製作	テスト		内資評価基準（参考） 影響度 設計	製作	テスト	
改造・再構築特性	母体調査ツールの機能水準 (P16)	母体の調査ツールの機能水準の高低を評価する。 ・該当するツールなどの機能が以下の内容に当てはまる個数で評価する a.絞込み機能　b.モニタリング（ルート解析）機能 c.ドキュメント生成（リバース）機能 d.実機での稼動確認　e.データディクショナリー機能（データ定義統一） fその他調査ツールなど		－	－	－	－					
				－	-20	-10	-20	４個～５個の機能を具備した調査ツールがある				
				－	-10	-5	-7	２個～３個の機能を具備した調査ツールがある				
				－	0	0	0	１個の機能を具備した調査ツールがある				
				－	15	5	10	調査ツールがない				
	既存設計書具備状態	・改造/流用母体の設計書有無および設計書が存在した場合のメンテナンス状態が、改造作業（Ａｌｌ）に及ぼす影響を評価する		－	－	－	－					
				0	0	0	0	完全にメンテナンスされた設計書が存在する				
				3	10	10	5	メンテナンス履歴が存在する				
				5	20	20	15	初期設計書のみ存在し、メンテナンス個所が不明				
				10	30	30	25	既存設計書が存在しない				
	既存のテスト環境流用水準 (P15)	・既存のテストドキュメントおよびテストデータ（テスト環境を含む）の流用可能度合をテストフェーズ別に評価する。	法規制（個人情報保護等）で本番データのテスト流用制限により手間が掛かる、および既存データの残存バグ率を考慮する。 流用率＝生流用率×データ加工率×データ残存バグ率	－	－	－	－					
				－	－	-20	-30	５０％～７０％未満が流用できる				
				－	－	-10	-20	２０％～５０％未満が流用できる				
				－	－	-5	-5	１０％～２０％未満が流用できる				
				－	－	0	0	１０％未満の流用に留まる				
	リソース管理水準	ソースとロードモジュールのバージョン管理水準		－	－	－	－					
				－	－	－	－					
				－	－	0	0	バージョン管理の仕組みが構築されており、その運用が徹底されている				
				－	－	5	5	バージョン管理の仕組みが構築されているが、その運用が徹底されていない				
				－	－	15	15	バージョン管理の仕組みが構築されていない				
	既存母体品質（正確性）	・既存システムが正しく動作しない場合の生産性に及ぼす影響を評価する ・既存母体調査の精度およびテスト作業における障害解析生産性が低下する	⇒ 直前１年間のバグ件数により評価し、残バグ率に応じて影響度を評価する 既存データの残存バグ率を考慮する	0	0	0	0	改造/流用母体の残バグ≒0件/10KS				
				2	3	3	3	改造/流用母体の残バグ≒0.1件/10KS				
				3	5	5	5	改造/流用母体の残バグ≒0.3件/10KS				
				5	8	8	8	改造/流用母体の残バグ≒0.8件/10KS				
				8	10	10	10	改造/流用母体の残バグ≒1.5件/10KS				
	既存母体品質（解析性）	・既存システムの改造個所特定（改造設計）におけるソースコードの解析容易性を評価する ・下記標準化項目の遵守度合いで評価する a.ソースコメント　b.修正履歴　c.モジュールサイズ　d.その他規約 d.その他顧客規約事項（モジュール分割指針、IF文ネストレベル等）		－	－	－	－					
				0	0	0	0	すべて遵守されている				
				3	10	10	5	遵守項目が３項目				
				5	15	15	10	遵守項目が２項目				
				10	25	25	15	遵守項目が１項目以下				
	既存母体品質（環境適用性）	・現行システム資産を別環境への移植し改造する開発における、別環境への適用容易性を評価する ・適用容易な環境の個数により評価する。 a.ハードウェア　b.ＯＳ　c.ネットワーク　d.フレームワーク（ミドル） e.ＤＢ/ＤＣ　f.バッチ運用システム	⇒ 同一環境で改造する開発においては評価対象外 ※「言語」「画面設計」「プログラム技術」は開発対象規模として扱う。 バージョン管理は『リソース管理水準』に含める。	－	－	－	－					
				0	0	0	0	適用用意項目が６項目				
				3	10	10	5	適用用意項目が４～５項目				
				5	15	15	10	適用用意項目が２～３項目				
				10	25	25	15	適用用意項目が１項目以下				

規模環境変数（改造型固有）

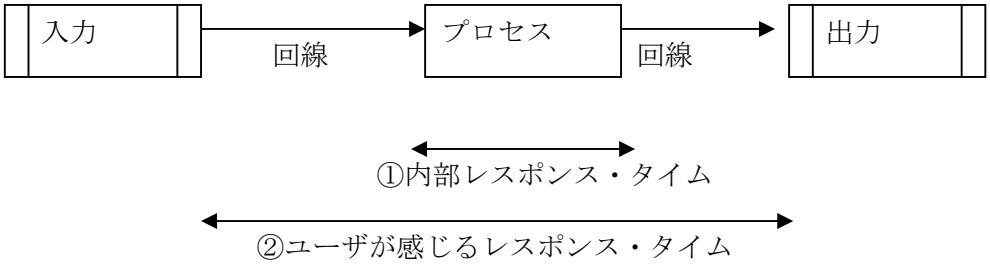
品質特性	副品質特性	評価の観点 内容	特記、特例事項	外資評価基準（見積プレゼン用） 影響度（%） 要件定義	設計	製作	テスト		内資評価基準（参考） 影響度 設計	製作	テスト	
機能性	（既存母体の）正確性	・改造/流用母体が正しく動作しない場合のテスト量（現行保証）に及ぼす影響を評価する	⇒ 直前１年間のバグ件数により評価し、残バグ率に応じてテスト量を増加させ品質向上を行う	－	0	0	0	改造/流用母体の残バグ≒0件/10KS				
				－	0	3	3	改造/流用母体の残バグ≒0.1件/10KS				
				－	0	5	5	改造/流用母体の残バグ≒0.3件/10KS				
				－	3	10	15	改造/流用母体の残バグ≒0.8件/10KS				
				－	5	15	25	改造/流用母体の残バグ≒1.5件/10KS				

付 11-2　レスポンス・タイムの種類

ユーザにとっては、応答時間は早ければ早いほど良い。しかし高速レスポンス・タイムを要求すればコンピュータやネットワークの資源を十分に用意しなければならないので、高価な資源費用を支払わねばならない。ここで、さまざまなレスポンス・タイムの種類があることを認識して、その要求をする必要がある。

レスポンス・タイムの種類とそれを得る対策の関係を、以下に整理してみる。

前提として整理すべきことは、レスポンス・タイムとはどの範囲を指すかの明記である。



ここで問うているレスポンス・タイムは上記②の、データを入力してからシステムで何らかの処理を行い、結果を入手するまでの時間である。この時間は通常はタイマーを手に測定可能であるので以降はこの定義でのレスポンス・タイムで議論を進める。

しかしながら日常コンピュータ・システムで自動的に把握可能なデータはコンピュータに入ってから出るまでの時間①である。運用データと検査用データにギャップがあることに注意したい。

レスポンス・タイムは、それを得るための環境にいくつかの種類がある。下記のどのレスポンス・タイムについての保証を求めているのかを RFP に明記することが必要である。つまり、「今回求めているレスポンス・タイムは、ケース X である。テスト環境はハードウエア、ネットワーク、テストデータ含めて発注者側が提供する」などの記述が必要となる。

図表付 12-1　レスポンス・タイムの種類

レスポンス・タイムの種類	説明	保証
1：開発マシーン環境 端末機は 1 台	開発会社のテスト環境 1 台の端末機でのレスポンス・タイム	A
2：開発マシーン環境 端末機は n 台	開発会社の端末機をつないでのテスト環境でのレスポンス・タイム、（n 台同時入力して最後の入力の応答時間）	A
3：本番環境 端末機は 1 台	本番環境ではあるが、他の入力はない場合のレスポンス・タイム	A、B

4：本番環境 端末機はｎ台（ｎ＜99)	本番環境で同種データｎ個同時入力した場合の最終テストデータのレスポンス・タイム	B
5：本番環境、端末機はｎ台（ｎ＜99)	本番環境で同種データｎ個同時入力し、干渉が起きる他のプログラムが背景にながれている場合の、最終入力のテストデータのレスポンス・タイム	B
6：本番環境、端末機の数は実稼働時と同じ(数千台以上のこともある)	本番環境で同種データｎ個同時入力した場合の最終テストデータのレスポンス・タイム（ｎはインターネットからの三桁/秒以上の入力件数を想定した場合) ただし同時に干渉が起きる他のプログラムが背景にながれている。最後の入力データに対するレスポンス・タイム	C

注)

A：プログラム、DB の作成方法の確認（保証者はプログラム作成企業)。ただしテスト機の指定は必要。

B：プログラム、DB の作成方法の確認（保証者は本番環境提供ベンダー)。ただしテスト環境・能力の指定は必要。

C：本番環境でのレスポンス・タイム保証。対策として GRID コンピューティングなどの活用も含む。

付 11-3　保守性について

保守性についての ISO の定義

保守性については、ISO／IEC 9126：2001 で以下のように定義されている[ISO01]。

- 解析性：故障や変更要求などを解析し、原因や修正が必要な部分を識別しやすいこと。または変更要求に対してどこを変えるべきかが分かりやすいこと（コードの理解のしやすさ）。
- 変更性：障害訂正などの変更のしやすさ（コード修正のしやすさ）。
- 安定性：障害訂正などの変更において、期待通りに正しく変更でき、かつ必要な箇所だけが変更されており、他の部分に影響が及ばないように作られている程度（修正による影響の少なさ）。
- 試験性：変更後の確認試験のしやすさ（修正したコードのテストのしやすさ）。

保守性を確保するための JUAS の提案

また JUAS は、保守作業を容易にするために、プログラムを以下のように作成することを提案している。

図表 12-2　JUAS の保守容易性実現のための手だて

1：ドキュメント、プログラムが理解・修正しやすい構造になっている。	(1-1) 機能構成が構造化、共通化されている。 (1-2) Hot module（注 1）は、特に外部テーブル化、共通ルーチン化などを図っておく。
2：ドキュメント、プログラムが理解しやすい表現になっている。	(2-1) 同一意味の単語は同一表現を用いる。 Ex：図書、書籍、出版物、本など (2-2) プログラムはシンプルに書かれている。 (2-3) コーディングスタイル、コメントの書き方を統一する。
3：ドキュメント、プログラムがテストしやすくなっている。	(3-1) Traceability（機能追跡性）が配慮されてある。 (3-2) 処理の中間結果がふんだんに残されている。

（注）Hot module：仕様から考えて、あるいはシステム再開発の場合は、過去の保守実績からみて、修正･追加が多いプログラムを Hot Module と呼ぶ。仕様変更の多いことが分かれば、その部分をさらに細分化し変更範囲を少なくしておく、個別の処理をプログラムに持ち込まず外部テーブル化してプログラマの手を煩わさずに、機能の修正を可能にする、などの対策が取れる。

12. USDM を使用した要求仕様書の有効性の確認

（例題・図書管理システムへの適用による効果確認）

この章の目的

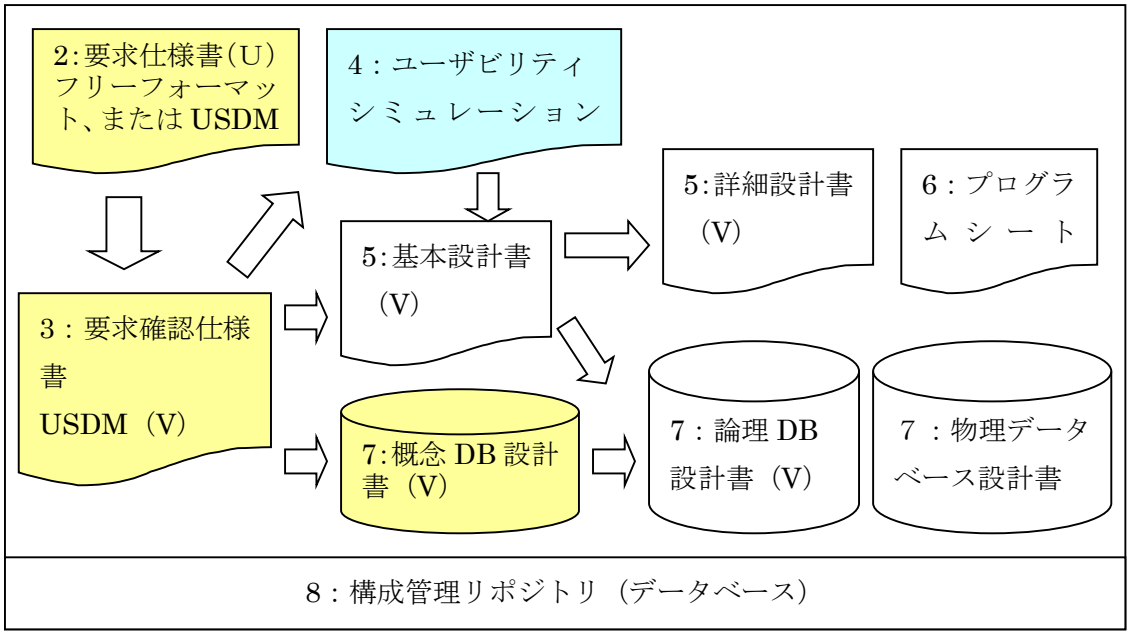
ユーザが準備する要求仕様書から、ベンダーの作成する要求確認書、さらに設計書へと設計が進むに連れて仕様が固まってくる。要求仕様書および要求確認仕様書に書かれてある情報をもとに下記のプロセス・フローに則り設計仕様の確認作業を進めることにより仕様書の充実、設計内容の緻密性が図れる。

本文に詳細に解説された内容が妥当であるか、課題は何かを確認するために、「企業内の図書管理システム」を事例にとり検証を試みた。

この章の最後に、この事例作成から得られて知見を示す。

１：プロセス・フロー

この例題試行で前提とする情報システム開発のプロセス・フローを、図表 12-1 に示す。



図表 12-1　この例題試行で前提とするプロセス・フロー

２：図書システムの要求仕様書

要求仕様書とは、「これから開発するソフトウェアの作業のゴールとしての要件を明らかにするものであり、顧客や開発関係者間でこれから作るものについての合意書である」と、清水氏は定義している[SIM05]。

なおこの章に記述する要求仕様書は USDM 表記法ではなく、フリーフォーマットで書か

れている。この場合の対応については、次の第 13 章で議論する。

（2-1）図書システムの要求仕様の内容

ユーザ側の図書管理に詳しいユーザが書いた仕様を以下に示す。

ここまでユーザが書いてくれるケースは、まれと言っても良い。

―――――――――――― ユーザ仕様 ――――――――――――

目的：

① 会社の図書館の書籍情報を共有化し有効に活用するために、人手の管理負荷を最小限にすること。

② 社員が必要とする書籍を探す負荷、貸し出しを申し込む負荷、返却する負荷を最小限にしかつ迅速に必要書籍を入手できること。

③ 書籍は図書コード別の棚に分けて収納するので特にシステムでの管理はしない。

前提条件：

（1）ある会社の資料室の図書の貸出と保管のシステムを想定する。

（2）資料室には、開架式の書棚が置かれ、１万冊の蔵書と各種雑誌が保管されている。

（3）社員が貸出を受けるには、

① 資料室に直接出向き、書籍を閲覧した上で、その場で貸出を受けることができる。

② または各自の端末から、貸出申請を行う。
資料管理者が、午前・午後１度ずつ申請状況を確認し、申請のあった図書を各申請者に対して、社内便・宅急便を使用して送達する。

③ 図書は一人について 10 冊まで、４週間以内で貸出を受けることができる。

（4）今回の研修にあたっては、このような資料室を運営する資料管理者のユーザビリティを配慮した図書管理システムの設計を行う。

（5）資料管理者が図書管理システムを使用して行う業務は以下のとおりである。

① 新規図書の登録

・書籍属性登録

・管理用シールの出力（書籍の固有番号：(ISBN)＋(nn)）を図書に貼る。

② 図書の確認、登録内容の修正、廃棄

※検索機能必要

③ オンライン貸出処理

・社員がオンラインで申し込む。

・最大 10 冊／人まで、最長４週間まで貸し出す。

・社員証（社員 ID）を基に本人であることを確認する。

④ 資料室貸出

・来訪者用：手渡し

・オンライン申請者用：本人部署に送付する（兼務者の場合は複数勤務地がある)。

⑤　返却処理

・管理用シールの番号で返却処理する。

⑥　延滞催促

・期限切れの者に督促メール発行する。

⑦　書籍購入（今回はシステム化対象としない)。

――――――――――　ユーザ仕様・終わり　――――――――――

システムを作成するには上記条件だけではまだ準備不足であり、以下の情報を IT 部門の SE が付加した。

――――――――――　SE によるユーザ仕様への追加　――――――――――

３：前提追加条件

（3-1）書籍のデータ属性（以下の範囲に限定する）

①　書名(100 文字、全角換算 50 文字)

②　著者・編者・訳者(100 文字、全角換算 50 文字)

③　出版社名(50 文字、全角換算 25 文字)

④　発行年月日(8 桁)

⑤　ISBN(10 桁)

※なお図書によっては雑誌＋論文集など ISBN コードがないものがあるが、その場合は自社コードをつける。

⑥　分類(20 文字、全角換算 10 文字)

⑦　目次（フリー)

⑧　イメージ(表紙画像)(別ファイル：サイズ固定、300KB)

⑨　備考(200 文字、全角換算 100 文字)・・・キーワード等を登録

（3-2）図書館属性

①　20 種類の固定の棚に分けられて保管されている。

②　9：00 開館、17：00 閉館

③　検索用端末 3 台設置（端末アクセス数 50 人／日程度）

④　利用者数 2,000 人程度

⑤　書籍数 50,000 冊

（3-3）社員のデータ属性（以下のデータが人事 DB から取得できる）

①　社員 ID（8 桁）

②　所属(20 文字、全角換算 10 文字)

③　役職(20 文字、全角換算 10 文字)

④　所属住所(100 文字、全角換算 50 文字)

⑤　郵便番号（7 桁）

⑥　電話番号（15 桁）

⑦　e メールアドレス（半角 50 文字）

＊兼務扱いがあり、②から⑦までは 3 個まで拡張情報を持つ。

―――――――――― SE によるユーザ仕様への追加・終わり ――――――――――

４：図書管理システム・ユースケース

図書管理システムの資料管理者の作業全体のユースケース図を、図表 12-2 に示す。

図表 12-2　ユースケース図

５：図書管理システム・ユースケース記述

資料管理者の作業

UC1：新規書籍の登録

・概要　　新規に購入した書籍の情報登録を行う。

・アクター　　資料管理者

・動作詳細　　登録内容の確認画面で確認したあとに、保存もしくは再度修正を行う。登録完了後、書籍の管理用シールを印刷し、それを書籍に貼る。管理用の番号は、同一書籍が複数存在する前提で、ISBN+nnn の番号をつける。

・例外時の動作　ISBN が存在しない報告書や雑誌については、何らかのコードをつけて管理する。

UC2：書籍の管理

・概要　　登録書籍の情報の編集、確認、廃棄などの処理を行う。

・アクター　　資料管理者

・動作詳細　　検索画面で書籍を検索し、検索結果を表示する。
当該書籍の詳細を表示し、情報の編集を行う。
廃棄書籍については、ステータスを廃棄とする。

・例外時の動作　検索結果の該当がない場合は、エラーメッセージを表示する。

UC3：オンライン貸出処理

・概要　　社員は自分の保有する PC から貸出申請を行う（この画面は省略)。

・アクター　　資料管理者、社員

・動作詳細　　貸出申請リストを表示し、印字する。
（資料管理者は、貸出書籍リストをもとに書棚から貸出書籍を抜き出す。）
書籍の管理用コードをもとに貸出処理を行い、貸出票を印字して、書籍とともに申請者に送付する。

UC4：資料室貸出処理

・概要　　社員は資料室を訪問し、自分の希望する図書を、貸出受付に持ち込む。
資料管理者は、社員 ID と書籍の管理用コードをもとに貸出を行う。
貸出の際に、貸出票を作成し、書籍と一緒に渡す。

・アクター　　資料管理者、社員

・動作詳細　　図書貸出にあたって、社員 ID と書籍の管理コードをもとに、貸出処理画面で貸出処理を行い、貸出票を印刷して書籍とともにその場で渡す。

UC5：返却処理

・概要　　社員からの返却書籍の処理をする。

・アクター　　資料管理者

・動作詳細　　貸出中のリストを表示し、返却書籍についての返却処理を行う。

UC6：督促処理

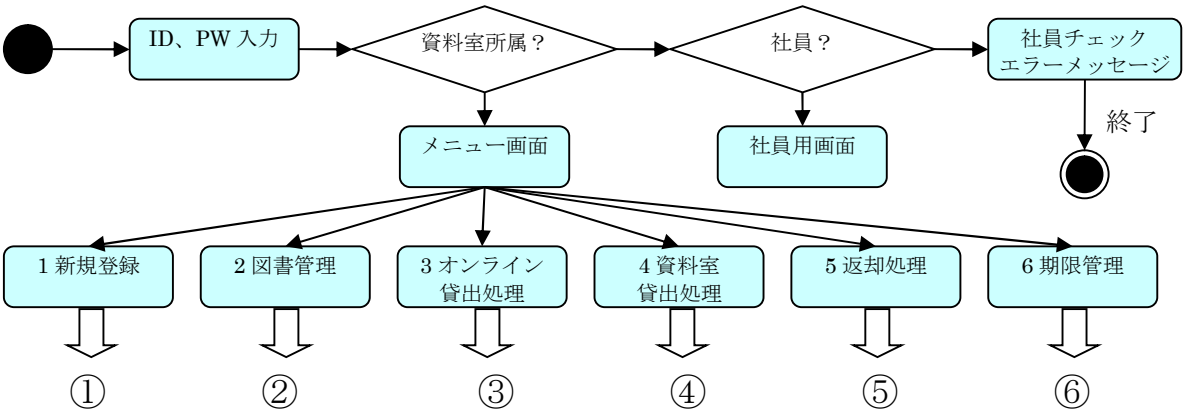
・概要　　貸出期限切れのリストをもとに督促メールを出す。

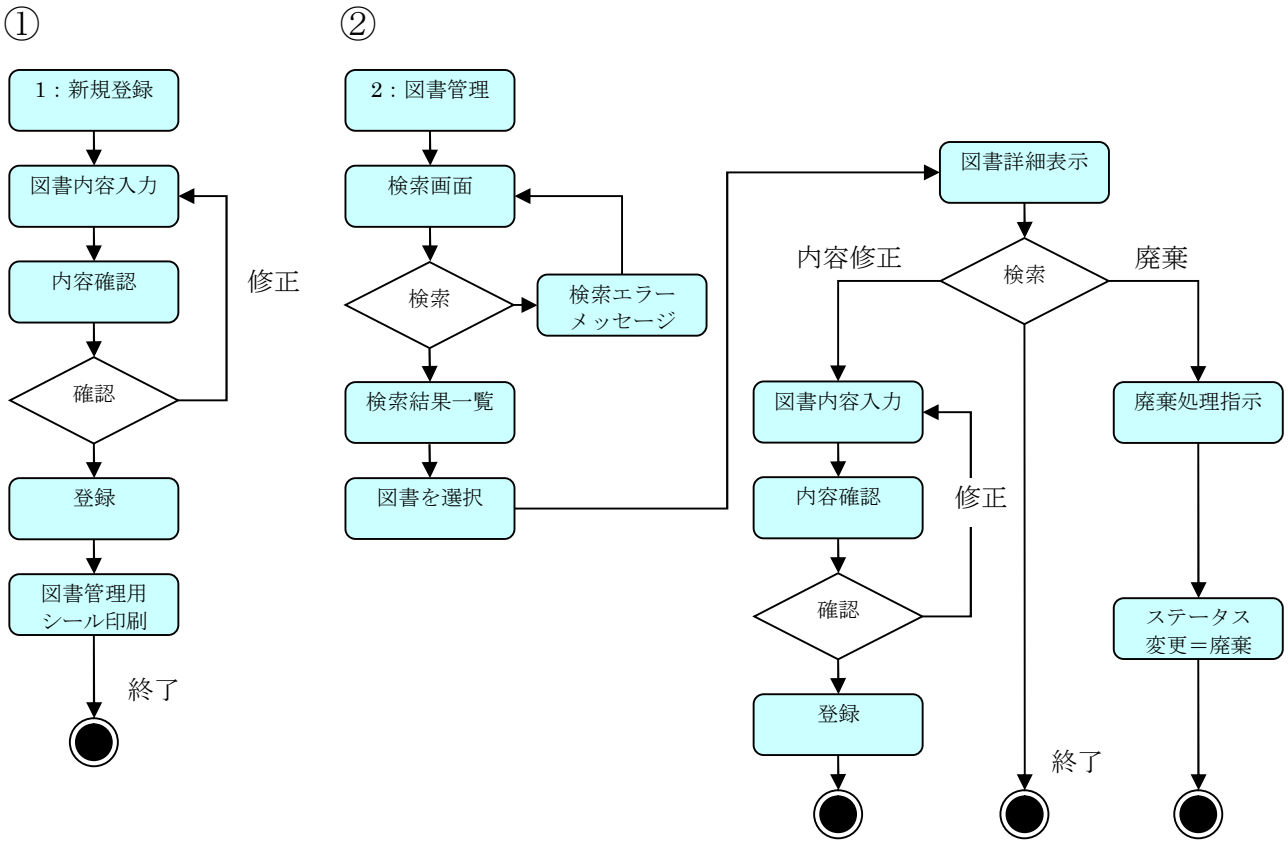
・アクター　　資料管理者

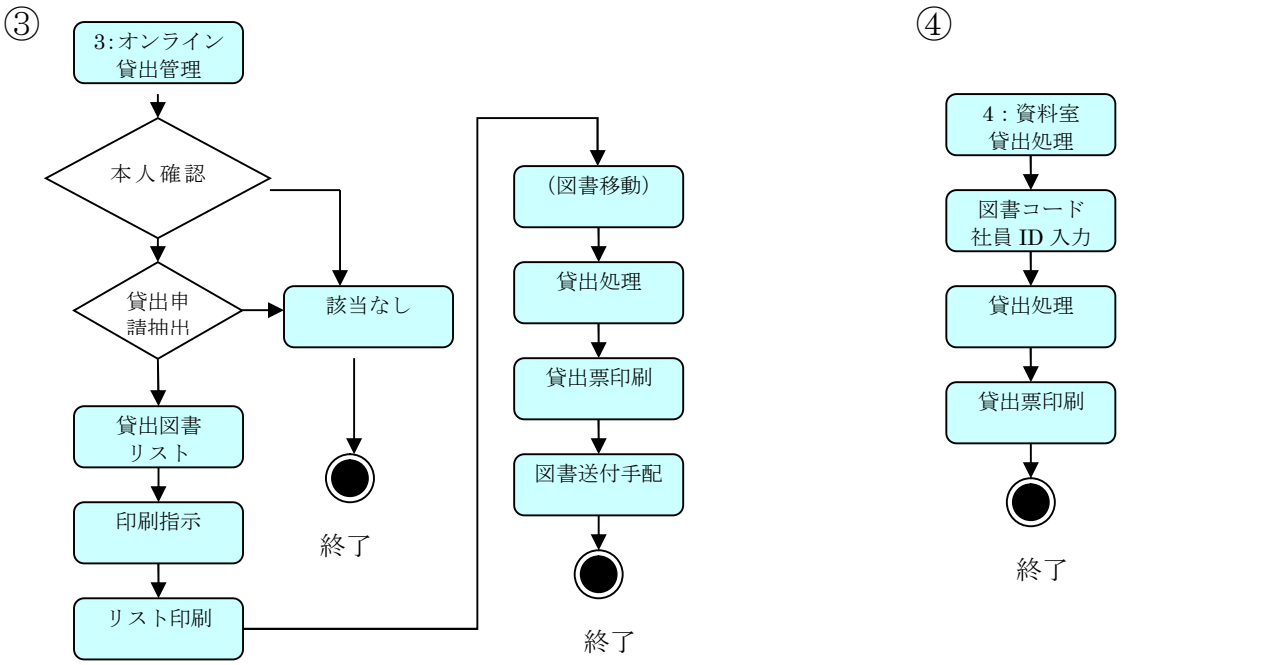
・動作詳細　　貸出期限切れのユーザのうち指定したものに督促メールを出す。督促メールはあらかじめ用意したテンプレートをもとに自動発行する（督促履歴機能などは省略する）。

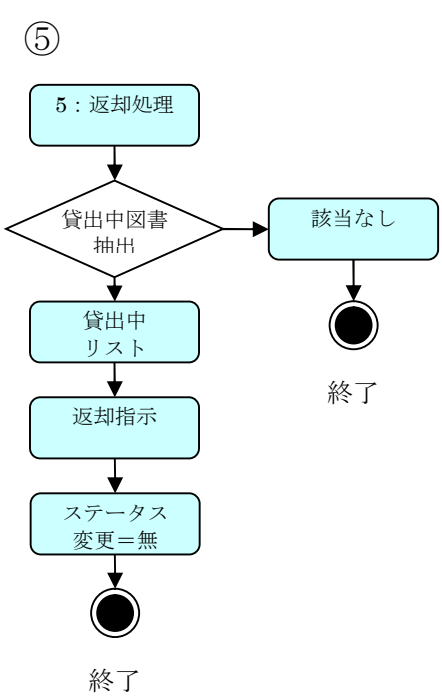
５：業務フロー

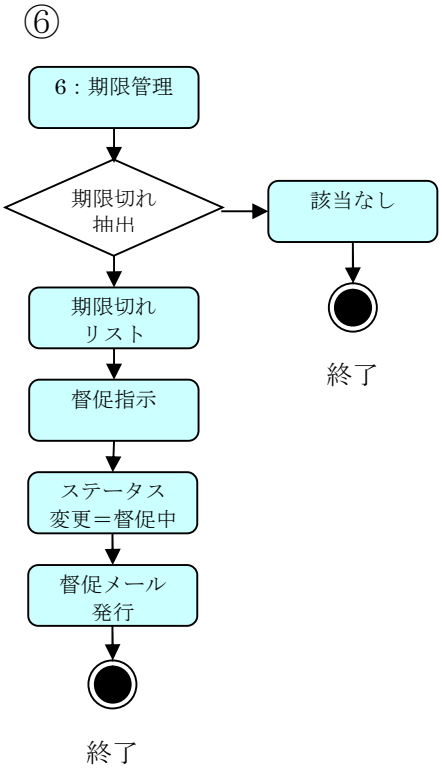
この作業に関わる画面の流れを、以下に示す。




①
1：新規登録
図書内容入力
内容確認
確認
修正
登録
図書管理用
シール印刷
終了
②
2：図書管理
検索画面
検索
検索エラー
メッセージ
検索結果一覧
図書を選択
図書詳細表示
内容修正
検索
廃棄
図書内容入力
内容確認
修正
確認
登録
終了
廃棄処理指示
ステータス
変更＝廃棄


③
3:オンライン
貸出管理
本人確認
貸出申
請抽出
該当なし
終了
貸出図書
リスト
印刷指示
リスト印刷
(図書移動)
貸出処理
貸出票印刷
図書送付手配
終了
④
4：資料室
貸出処理
図書コード
社員 ID 入力
貸出処理
貸出票印刷
終了





６：新規図書の登録機能の定義

①ユーザからの要求仕様

UC1：新規書籍の登録

- 概要　新規に投入した書籍の情報登録を行う。
- アクター　資料管理者
- 動作詳細　「新規書籍登録」画面で、書籍属性を登録する。
  登録内容の確認画面で確認した後に、保存もしくは再度修正を行う。
  登録完了後、書籍管理用シールを印刷し、それを書籍に貼る。
  管理用の番号は、同一書籍が複数存在する前提で「ISBN＋nnn」の番号をつける。
- 例外時の動作　ISBN が存在しない書籍や雑誌については、ユニークな類似コードをつけて管理する。

＜書籍属性＞

① 書名(100 文字・全角換算 50 文字)

② 著者・編者・訳者(100 文字・全角換算 50 文字)

③ 出版社名(50 文字・全角換算 25 文字)

④ 発行月日(８桁)
⑤ ISBN (10 桁)　なお雑誌，社内発行論文集など ISBN コードがないものがある。
⑥ 分類(20 文字・全角換算 10 文字)
⑦ 目次（フリー）
⑧ イメージ(表紙画像)（別ファイル：サイズ固定、300KB）
⑨ 備考（200 文字、全角換算 100 文字)・・・キィワードなどを登録
ユーザから上記のような機能要求があった場合の機能仕様の確認は図表 12-3 のごとく展開される。

このような与えられた条件で、ユーザビリティの設計をお願いした。

７：得られた知見

① 機能仕様の提示

ユーザは概略機能仕様を提示してくれた。このレベルは一般的に見れば相当に良く出来た仕様書であったが、システム化の目的、目標、前提条件（データ量、利用者数、端末台数、など）の条件提示は不十分であったので、レビュアー（IT 部門の SE）が追加、補足した。

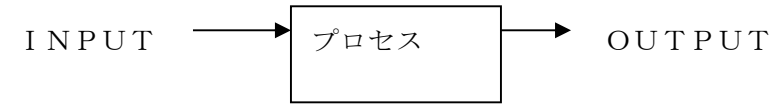
この情報が設計に重要な影響を与えるので注意を要する。

要求仕様書に記載すべき情報、機能要求、非機能要求、データベース情報、ユーザビリティへの希望は明確に定義せねばならない。

② 非機能仕様とは何かが、議論になった。

レスポンス・タイム、セキュリティ、ユーザビリティは機能ではないかとの意見も出されたが、以下の定義によることにした。

・機能要求の定義



何を入力し、加工し、何を出力するのか？が定義できるものは機能であり、それ以外の要求仕様は非機能要求である。

・非機能要求

レスポンス・タイム、バッチ処理の時間、保守性、信頼性、などを要求する

③ 機能仕様の明確化　USDM 法の活用

仕様と理由を分離して記述することにより、

① 何故その仕様が必要なのかを、正しく伝えることが出来る

② 仕様のトレースが可能になる

ことが明らかになった。

なおUSDM方式に入る前にユースケース図あるいは機能階層図の機能単位をどのようにまとめるのかは、後々の仕様分解作業に影響を与えるので吟味が必要である。

実例でも抽象化、標準化を考えた機能の区分、括り方と機能の記述方法はベテランと未熟者の差が大きく出た。

USDM方式も訓練とSEのレベルアップへの努力が必要である。

この中の書籍登録（UC1）の機能の部分をとり、清水氏に仕様の整理をお願いした。この結果を図表12-3に示す。

またこの要求仕様についての清水氏のコメントを、章末に付12-1として添付する。

④ 概念データベースの設計

要求確認仕様書を基に概念データベース設計をする。仕様の不整合や漏れを発見する手段として“自然言語” から“図面言語”への変換を試みる手法は、仕様の不足している部分、曖昧な部分が解明され仕様が一段と明快になった。

この図面による要求の分析は非常に有効であるが、やはり訓練と経験を必要とする。

図書管理システム全体の概念データモデルを、図表12-4に示す。

⑤ ユーザビリティ（シミュレーション）

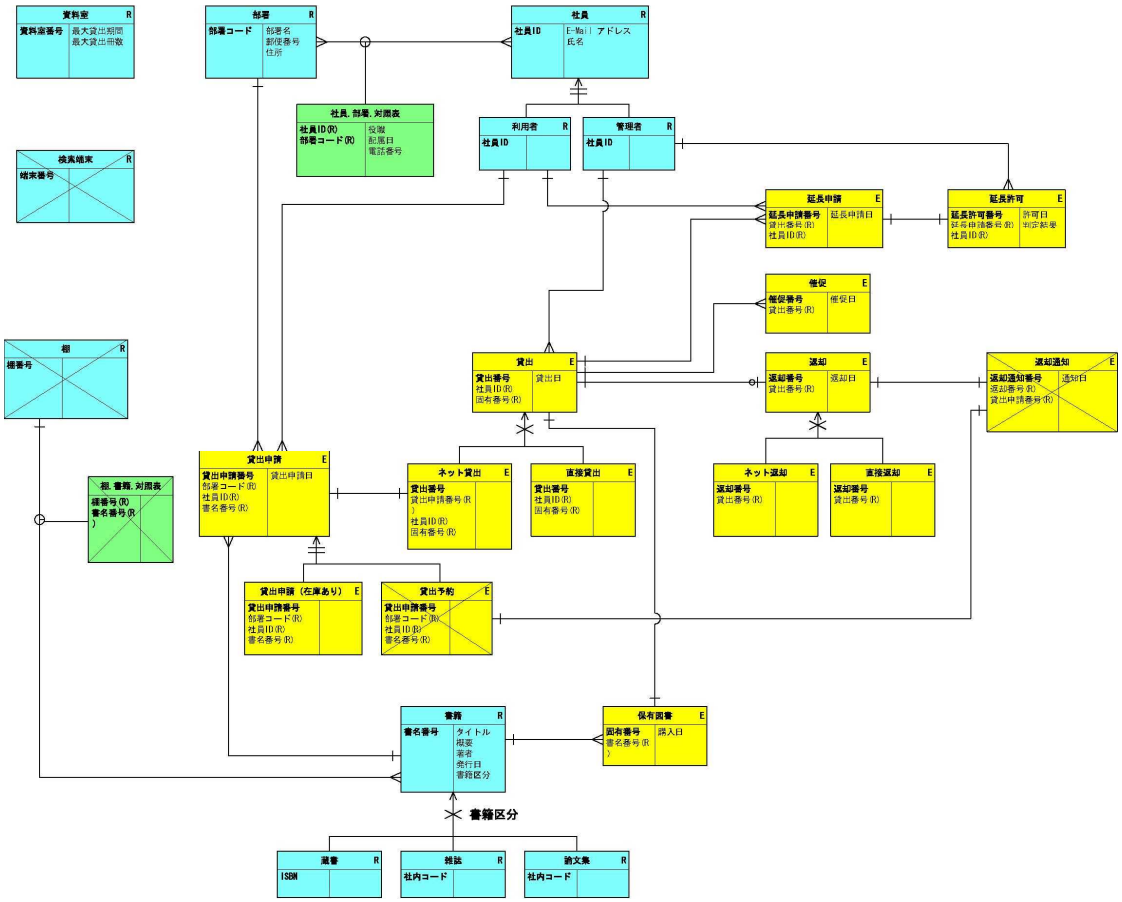
ペーパー・プロトタイプ・アプローチ、スクリーン・プロトタイプ・アプローチ、オンライン・プロトタイプ・アプローチの3手法を使い分け、簡易なシミュレーションを実施することにより、画面フローと画面設計がこの段階で確認される。

入力担当者のユーザと設計者がプログラム作成前に、画面の操作性を確認できるので、最終段階の総合テスト後に「使いにくいから画面を修正して欲しい」等の苦情が発生しないことになる。この効果も大きい。

このユーザビリティをどのフェーズで実施し、確認するかはアプリケーションシステムの性格によって異なるが、できるだけ開発の初期段階に実施するのが望ましい。

要求	ＵＣ1	新規書籍購入、自社発行書籍・雑誌の情報を登録し、必要なら書籍に添付するバーコードのカードを印刷する		
	理由	書籍を区別するため		
	説明			
	要求	ＵＣ1-１	ISBNコードが付いているときは、ISBNコードで登録する	
		理由		
		説明		
		＜ＩＳＢＮコードの入力＞		
	□	ＵＣ１－１－１	新規書籍登録画面を呼び出し、ＩＳＢＮコードをバーコードリーダーで読み、ＩＳＢＮコード欄（１０桁）に書き込む。	
	□	ＵＣ１－１－２	ＩＳＢＮコードが何らかの理由で読めなかった場合は、付いている数字を読み取り番号入力する。	
		＜既存登録の確認＞		
	□	ＵＣ１－１－３	ＩＳＢＮコードを基に、すでに同じ本がないかどうか確認する	
			ＩＳＢＮコードが同じで異なる書籍名が付いている場合は、ＩＳＢＮコードなしの扱いをする（ＵＣ１-２-１へ）	
		＜新規登録時の追番の付加と書籍情報の入力＞		
	□	ＵＣ１－１－４	新規登録の場合は「００１」から、今回登録する冊数だけ連続した追番をつける。	
	□	ＵＣ１－１－５	新規登録の場合、登録に必要な書籍属性を本から読み取って入力する 【説明】この操作は新規登録時のみ	
		＜既登録時の追番の付加＞		
	□	ＵＣ１－１－６	同じ番号の書籍が登録されている場合は、現在の追番の最大値から今回登録する冊数だけ連続した追番をつける。	
	要求	ＵＣ1-２	ISBNコードが付いていないときは、書籍名から登録する	
		理由		
		説明		
		＜書籍名による確認＞		
		ＵＣ１－２－１	自社作成書籍管理ＤＢ（注1）から書籍名を入力して同じ書籍の登録があるかどうか確認する	
		＜新規登録時のコードの付加と情報の入力＞		
		ＵＣ１－２－２	自社発行分１桁＋社内コードを付ける	
		ＵＣ１－２－３	新しく００１番から今回登録する冊数だけ連続した追番をつける	
		ＵＣ１－２－４	書籍属性を、書籍から読み取り入力する	
		＜既登録時の追番の付加＞		
	□	ＵＣ１－２－５	同じ名称の書籍が登録されている場合は、現在の追番の最大値から今回登録する冊数だけ連続した３桁の追番をつける。	
	要求	ＵＣ1-３	書籍添付用カードを発行し本に添付する。	
		理由	貸し出し作業の効率化、書籍の確認のため	
		説明	本に１枚ずつ添付するもの	
		＜添付用カードの内容＞		
		ＵＣ１－３－１	自社発行分コード（１桁）、書名番号（7桁）と追番（３桁）	
			追番は、今回割り当てた番号	
		＜印刷枚数＞		
			今回登録時に入力した冊数だけ	
	要求	ＵＣ１－７	書籍番号カードの発行機能を持つ	
	理由	本の紛失後の紛失者による再購入、カードの破損などの対応を可能とするため		
	説明			
	要求	ＵＣ１－７－１	再発行したい書籍の先頭番号と最後の番号をISBNコード（自社発行コードの体系を含む）+追番で入力する	
			入力された番号の書籍が登録されている場合は、確認のためにその情報を表示する	
		ＵＣ１−7−3	指示された最初の追番から最後の追番まで印刷する	

図表 12-3　図書登録に関わる部分の要求仕様書（USDM 表記法による）



図表 12-4　実体関連図

（この実体関連図は、「T 字型 ER モデル」による。）

参考文献

[SIM05] 清水吉男著、「[入門+実践]要求を仕様化する技術表現する技術～仕様が書けていますか」、（株）技術評論社、平成 17 年.

付 12-1　例題の USDM 表記法についての清水氏のコメント

					備考欄	システム設計書	タスク１設計書	タスク２設計書	関数設計書	ソースファイル	ソースファイル	ソースファイル	ソースファイル４
										トレーサビリティ・マトリクス			
	要求	UC1	新規書籍購入、自社発行書籍・雑誌の情報を登録し、必要なら書籍に添付するバーコード用カードを印刷する										
		理由	書籍を区別するため										
		説明											
		要求	UC1-１	ISBNコードが付いているときは、ISBNコードで登録する									
			理由										
			説明										
			＜ＩＳＢＮコードの入力＞										
		□	ＵＣ１－１－１	新規書籍登録画面を呼び出し、ＩＳＢＮコードをバーコードリーダーで読み、ＩＳＢＮコード欄（１０桁）に書き込む。		３．４、６．２				関数１			
		□	ＵＣ１－１－２	ＩＳＢＮコードが何らかの理由で読めなかった場合は、付いている数字を読み取り番号入力する。		３．４、６．２				関数２、関数８			
			＜既存登録の確認＞										
		□	ＵＣ１－１－5	ＩＳＢＮコードを基に、すでに同じ本がないかどうか確認する		3.5		2.5			関数３		
				ＩＳＢＮコードが同じで異なる書籍名が付いている場合は、ＩＳＢＮコードなしの扱いをする（ＵＣ１-2-１へ）									
			＜新規登録時の追番の付加と書籍情報の入力＞										
		□	ＵＣ１－１－8	新規登録の場合は「００１」から、今回登録する冊数だけ連続した追番をつける。									
		□	ＵＣ１－１－9	新規登録の場合、登録に必要な書籍属性を本から読み取って入力する 【説明】この操作は新規登録時のみ			4.2				関数４		
			＜既登録時の追番の付加＞										
		□	ＵＣ１－１－12	同じ番号の書籍が登録されている場合は、現在の追番の最大値から今回登録する冊数だけ連続した追番をつける。									
		要求	UC1-2	ＩＳＢＮコードが付いていないときは、書籍名から登録する									
			理由										
			説明										
			＜書籍名による確認＞										
			ＵＣ１－２－１	自社作成書籍管理ＤＢ（注1）から書籍名を入力して同じ書籍の登録があるかどうか確認する									
			＜新規登録時のコードの付加と情報の入力＞										
			ＵＣ１－２－4	自社発行分１桁＋社内コードを付ける									
			ＵＣ１－２－5	新しく００１番から今回登録する冊数だけ連続した追番をつける									
			ＵＣ１－２－6	書籍属性を、書籍から読み取り入力する			5.3				関数４		
			＜既登録時の追番の付加＞										
		□	ＵＣ１－２－10	同じ名称の書籍が登録されている場合は、現在の追番の最大値から今回登録する冊数だけ連続した３桁の追番をつける。									
		要求	UC1-３	バーコード用カードを発行し本に添付する。									
			理由	貸し出し作業の効率化、書籍の確認のため									
			説明	本に１枚ずつ添付するもの									
			＜添付用カードの内容＞										
			ＵＣ１－３－１	自社発行分コード（１桁）、書名番号（7桁）と追番									
				追番は、今回割り当てた番号									
			＜印刷枚数＞										
				今回登録時に入力した冊数だけ									
		要求	UC1-7	バーコード用カードの発行機能を持つ									
				本の紛失後の紛失者による再購入、カードの破損な									
			説明										
			ＵＣ１－7－１	再発行したい書籍の先頭番号と最後の番号をISBNコー	を含む）+追番で入力する								
				入力された番号の書籍が登録されている場合は、確認のためにその情報を表示する									
			ＵＣ１－7－3	指示された最初の追番から最後の追番まで印刷する									

元はUC1-4 の要求ですが、UC1の機能と相対する機能と判断したため、上位要求に配置しています
ＵＣ２は他の機能（貸し出し処理？）で使っていますので、元のままにしました。
（清水）この位置では、「UC1]の要求（＝図書の登録処理）の配下の要求になります。

「ＵＣ１－１－9」が「タスク１設計書」の「4.2」章で扱われており、さらに、ソースファイル２の「関数４」で実現していることを表しています

同じ関数で実現している状態を表しています

コードが書かれた後で、「ＵＣ１－１－9」の仕様が変更された時、設計書やソースコードの該当箇所がここから把握できます。
さらに、ソースファイル２の列を探すと、同じ関数が「ＵＣ１－２－6」の仕様実現にも関わっていることが分かります。この方法で仕様変更で影響を受ける仕様や、変更すべき箇所の把握

「８章」の「トレース方法-２」で書かれているように、仕様に付けられた番号を設計書やプログラム内に付けていく方法は現実的ではありませんし、役に立つ場面は限られます。
しかしながら、個々の仕様を扱っている設計書の「章・節番号」や、ソースファイルの関数名をマトリクスに書き込むことは、難しいことではありません。そしてこの方法によって、「要件管理」の中での仕様変更の対応で求められている「双方向追跡マトリクス」を実現します。

13. USDM 表記法の適用方法

この章の目的

この報告書では、9 章までと 11 章で一貫して、要求仕様書はユーザ企業がその責任で作成し、ソフトウェア・ベンダーに提示するものとして記述してきた。また情報システムの開発手順も従来からのウォータフォール型、つまり要求仕様書の作成から始まり、情報システムの分析からシステムの設計、実装につながるものを想定してきた。

しかし情報システムの作り方は、この報告書でこれまで前提にしてきたものばかりではない。例えばユーザビリティ研究会が研究を進めてきた JUAS-UCD モデルでの開発手順は、第 10 章で述べたとおりこれまでのウォータフォール型とは異なっている。

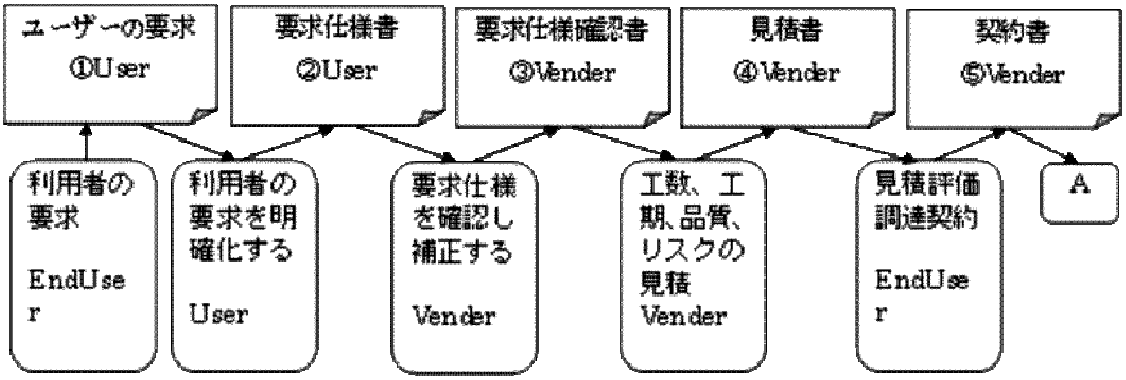
この章では、我々が研究してきた USDM 表記法を中心とする要求仕様の記述方法が、いくつかの情報システム開発の手順の中でどのように適用できるのかを考察しておきたい。

ただしその考察の前に、日本情報システム・ユーザー協会（JUAS）としてのこの研究に関わる 3 つのことを確認しておきたい。

要求仕様書（RFP）に記述するべき内容

最初の確認事項は、要求仕様書（RFP）に記載するべき内容についての確認である。

JUAS が想定する情報システム開発作業の流れは、図表 1-8 に示した。その一部を図表 13-1 として再掲する。



図表 13-1　情報システム開発作業の流れ

つまり要求仕様書（RFP）とは、ユーザがベンダーに利用者の要求を明確にした文書として提示し、それに基づいてベンダーはその内容を確認し、補正するものとして「要求仕様確認書」を作成して、ユーザにフィードバックすることになる。

この RFP には、次の 4 つの内容が記載されていなければならない。

- その情報システムが持つべき、機能についての要求
- その情報システムが持つべき、非機能についての要求

- その情報システムに関わるデータについての記述
- ユーザビリティについての記述

我々が定義した要求仕様書は、ユーザビリティについての記述方法が必ずしも充分とはいえない懸念はあるものの、これらを網羅しているということができる。

ビジネスシステム定義との関連

JUAS ではこれまで、情報システムの構築に関わる研究会を毎年継続して実施してきた。その 1 つに、平成 15 年度に実施したビジネスシステム定義に関する研究会があり、ここで情報システムの構築時に必要な 10 種類のドキュメントを挙げている。

その 10 種類とは、以下のものを指す[JUA04]。

① ビジネス機能構成図
② ビジネスプロセス関連図
③ 業務流れ図
④ 機能情報関連図
⑤ 業務ルール定義書
⑥ 個別業務処理定義書
⑦ 画面・帳表一覧
⑧ 画面・帳表レイアウト図
⑨ データ項目適宜書
⑩ 運用操作要件書

この中の「⑥個別業務処理定義書」には、「個別業務の入出力と業務ルールの関係を表現」することになっている。これには要求仕様書の機能要求が該当することになる。その意味で、我々の要求仕様書は、この JUAS のビジネスシステム定義の一部を詳細化したものと捉えることができる。

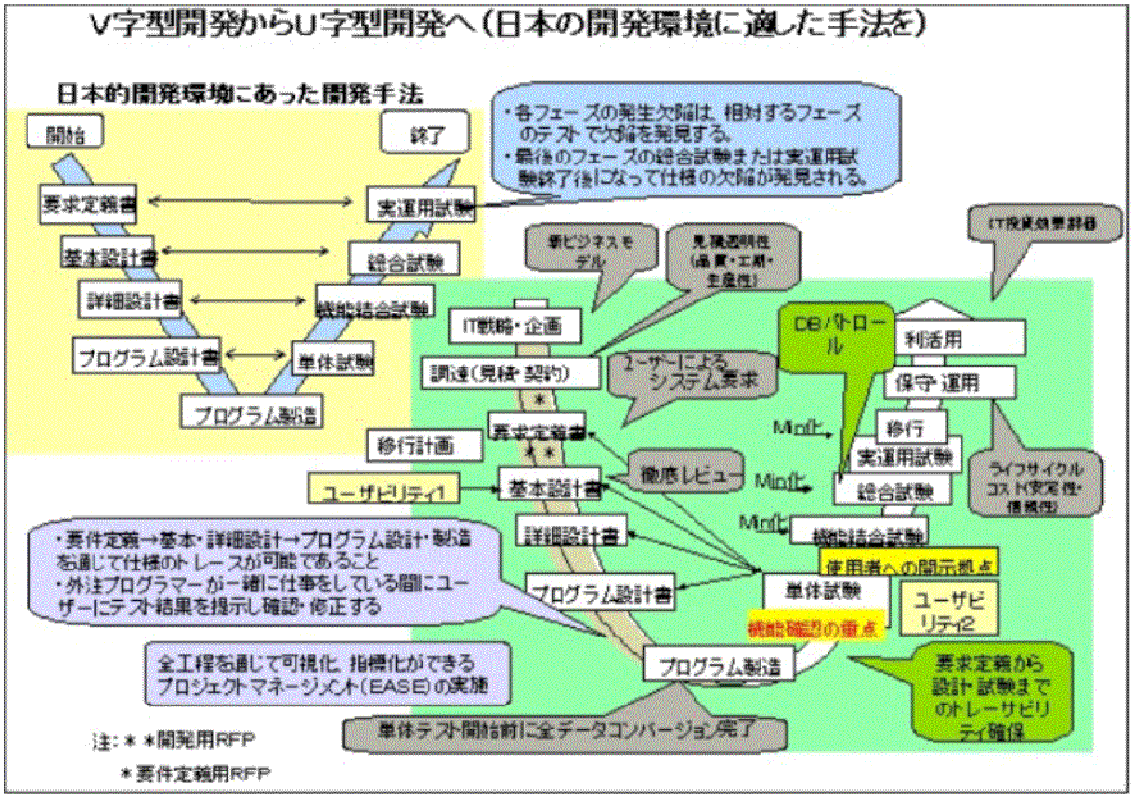
ちなみにこのビジネスシステム定義では、「⑥個別業務処理定義書」の記述は HIPO とフローチャートで行うことになっている[JUA04]。

U 字型開発法との関連

さらに JUAS では、情報システム開発の手順として U 字型開発法を提案している。U 字型開発法は図表 4-2 に提示した。それを図表 13-2 として再掲する。

U 字型開発法では、ユーザによるシステム要求は「要求定義書」という表現で記述されているが、これが我々の「要求仕様書」である。

したがって我々の要求仕様書は、この U 字型開発とも、整合性がとれていることが確認できた。



図表 13-2　U 字型開発モデル（[JUA05] より）

USDM 表記法の適用方法

3 つの確認が終わったところで、我々のこの書の本来の目的である「USDM 表記法の適用方法」について考えたい。換言すればこれは、いくつかの開発モデルの中で我々の USDM 表記法を核とした要求仕様の記述方法が、要求仕様書、あるいは要求仕様確認書などとして位置づけできることの確認といっても良い。

ここで情報システムの開発モデルとして、以下の 3 つを想定してみたい。

① USDM 表記法に基づいてユーザ企業が要求仕様書を書くケース

② ユーザは要求仕様書をフリーフォームで記述し、ベンダーが要求仕様確認書として USDM 表記法を使用するケース

③ ユーザビリティ研究会が提案する JUAS-UCD モデルに基づいて開発を行うケース

「①USDM 方式でユーザが記述するケース」は、まさに我々が想定して研究を進めてきた方式である。もう一つの「②ベンダーが USDM で確認するケース」は、我々は第 12 章の図書システムの例題で採用した方式である。最後の「③JUAS-UCD モデル」は、ユーザビリティ研究会が提案している方式である。

この 3 つの方式の特徴と関連性を、図表 13-3 としてまとめる。

図表 13-3　3 つの手法の特徴と関連性

活用の方式	①USDM 方式でユーザが記述するケース	②ベンダーが USDM で確認するケース	③JUAS-UCD モデル
特徴	ユーザが USDM 形式で要求を記述し、ベンダーが確認・補正する。	ユーザはフリーフォーマットで要求を記述し、ベンダーがその結果の確認を USDM 表記法で行う。	ユーザビリティを重視するプロジェクトでは、現状分析から UCD を活用する。機能確認は USDM 方式で実施する。
適用プロジェクト	新規プロジェクト（ユーザへのトレーニングが必要)。	ユーザが USDM で整理できないプロジェクト。	ユーザビリティ重視のプロジェクト（高度なユーザであり、機能は USDM で記述する)。
適用結果	ユーザの要求仕様の変化が直接確認できる。	ユーザの要求の仕方の明確化から確認できる。	ユーザビリティ要求の仕方から、機能の展開方法までが確認できる。
U字型開発法との関係	整合性あり。	整合性あり。	整合性あり。
DB 設計	ベンダーが確認。	ベンダーが確認。	ベンダーが確認。
ユーザビリティ設計との関係	3 つのプロトタイプ手法（シミュレーションを含む）を選択・活用する。	3 つのプロトタイプ手法（シミュレーションを含む）を選択・活用する。	UCD 開発メソッドにより推進する。
作成ドキュメント	10 種（ただし機能の詳細記述は USDM 表記法による)。	10 種（ただし機能の詳細記述は USDM 表記法による)。	ペルソナ、シナリオ、ワークタスク、機能仕様書、ユーザ・インタフェース仕様書が加わる（ただし機能の詳細記述は USDM 表記法による)。

まとめ

この研究会の参加企業が、要求仕様書を作成する前にどのような作業を行っているかを紹

介し合った。その結果、以下に述べるような方式で仕事の流れを記述してから要求仕様書を作成していることが明確になった。しかしその方式や作成する資料はそれぞれ異なっていることが明らかになった。

- 業務フロー図を作成する。
- PFD（Process Flow Diagram）を作成する。
- 業務コラボレーション図を作成する。

またこの報告書の中で明らかにしたように、実体関連図にもいくつかの記述の仕方があり、いずれも甲乙は付けがたい。この要求仕様書作成の後に続く基本設計、詳細設計の作業もまた各社各様の標準化が既になされており、そのやり方が異なっているのが現状であろう。

我々は、その現実をそのまま受け入れたい。つまり各社各様で進めている情報システム構築の方式を基本的にはそのまま継承し、その中で要求仕様を記述する部分には USDM 表記法を取り込む方式を提案したい。

また第 1 章と第 2 章で我々は、要求仕様書に関わる様々な問題点があることを明らかにした。最大の問題点は、「要求仕様書に仕様が記述されていない」というものだったが、今回の研究で上記のものを含む多くの問題を解決できたと自負している。

参考文献

[JUA04] 「エンドユーザによるビジネスシステム定義の進め方 平成 15 年度ビジネスシステム定義研究プロジェクト報告書」、（社）日本情報システム・ユーザー協会、平成 16 年 4 月.

[JUA05] 「システム・リファレンス・マニュアル（SRM）」、（社）日本情報システム・ユーザー協会、2005 年 9 月.

14. 残された課題

残された課題

今年度のこのプロジェクトによって、要求仕様書の作成に関わるいくつかの問題について決着を付けることができた。

しかしそれに関連して、次のようなテーマがまだ問題として残されていると考える。いずれも次年度以降の課題として、順次取り上げてゆきたい。

1. 基本設計書と詳細設計書の作成に関わるもの
2. データベースの設計に関わるもの
3. 構成管理のリポジトリに関わるもの
4. ソフトウェア・ベンダーとの契約に関わる問題
5. テストの実施に関わる問題

以下でこれらの問題について、今少し掘り下げておきたい。

１．基本設計書と詳細設計書に関わる問題

USDM 表記法では、機能仕様に「理由」を付加しながら順次詳細化してゆく。

この「理由」を書くことにより、「何故このような仕様にしなければならないか」を設計者が順次確認できることになり、設計内容の是非を再度考え直すので仕様が正確になり、かつ分り易くすることができる。

しかし仕様記述の標準化が企業によって異なるので、詳細をどこまで書くのかがこれからの検討課題になる。

優れた標準化案が出来れば、設計記述は簡単になる。ビジネス・システムにおけるこの局面の標準化の検討は、今後の課題である。

ユーザビリティの設計フェーズへの取り組み方も、実施結果を見ながら適切なフェーズとアクションを結びつける必要がある。

２．データベースの設計にかかわるもの

①．データの属性について

データベースに保管するデータの属性（データ名、桁数、英数字、意味など）は、ユーザが提供しなければならない。画面／帳票により項目を拾い出して、属性を整理する必要がある。

ここで、同じ意味のデータは同じ名称に統一しておくことが重要である。この作業をここでは、「データの標準化」と呼んでおく。

②．概念データベースから論理データベース、物理データベースへの展開

要求仕様書作成段階で準備された概念データベースの必要部分のみを切り出し、設計段階

で論理データベースに加工しなおす。さらに実装段階で、物理データベースに展開される。この具体的な手法の紹介は DOA などすでに有識者の出版物があるので、内容の紹介はそちらに依存したい。

しかし前述の「データの標準化」を含めたこれらの作業についての必要性の認識は、今ひとつのように思える。この啓蒙は、今後の課題の 1 つである。

３．構成管理のリポジトリ（データベース）に関わるもの

構成管理のリポジトリ（ソフトウェア開発用のデータベース）が望ましい形で作成されると、総てのドキュメントは仕様番号つきで、リポジトリに構造化された形で保管される。

したがってこれによって、要求仕様書から設計書、プログラムに至る機能仕様のトレーシングが可能になる。この有効性は、開発局面は勿論のこと、保守段階を通しても発揮されることになる。

最近の情報検索技術の進化は目覚しいので、それらの技術を併用した構成管理への挑戦が期待される。

４．契約条件について（納期、提出物、納入物など（参照：SLCP-JCF98 または 2007））

今回のプロジェクトでは、この項目について特に議論をしていない。

しかし経済産業省が平成 19 年度にまとめた「情報システムの信頼性向上のための取引慣行・契約に関する研究会」～情報システム・モデル取引・契約書～（受託開発（一部企画を含む）、保守・運用）＜第一版＞が別途あるのでそれを参考にし、必要があればこれをベースにして、ユーザとベンダー間で契約時に交わす項目、詳細度などについての議論を行いたい。

５．テストの実施に関わる問題

要求仕様書が適切に作成されると、テストシナリオの設定、テストケースの洗い出しが容易になり、結果としてテストの生産性とソフトウェアの信頼性が向上するといわれている。

ついては、今回検討した要求仕様書を基にして、テスト段階への展開を検討したい。併せて、U 字型開発手順で JUAS が提案している早期での業務仕様のテスト実施についても、明確な指針を提示したい。

付１．参考文献

[AKI04] 秋本芳伸、岡田泰子著、「若手 SE のための要求仕様のまとめ方」、（株）ディー・アート、2004 年.

[BOO99] グラディ・ブーチ著、オージス総研オブジェクト技術ソリューション事業部訳、「UML ユーザーガイド」、（株）ピアソン・エデュケーション、1999 年.

[BRO75] フレデリック・P・ブルックス，JR 著、滝沢徹他訳、「人月の親和 狼人間を打つ銀の弾はない 原書発行 20 周年記念増訂版」、アジソン・ウェスエイ・パブリッシャーズ・ジャパン（株)、1996 年.

[COC01] アリスター・コーバーン著、ウルシステムズ（株）監訳、「ユースケース実践ガイド-効果的なユースケースの書き方」、（株）翔泳社、2001 年.

[CRO79] フィリップ・B. クロスビー著、小林宏治監訳、「クオリティ・マネジメント：よい品質をタダで手に入れる法」、日本能率協会、1980 年.

[DAV05] アラン・M・デービス著、萩本順三他監修、高嶋優子訳、「成功する要求仕様失敗する要求仕様」、日経 BP 社、2006 年 11 月 6 日.

[HUK05] 日本情報システム・ユーザー協会編、福田修著、「SE を極める 仕事に役立つ文章作成術」、日経 BP 社、2005 年.

[IEE98] IEEE-SA Standards Board, “IEEE Recommended Practice for Software Requirements Specifications IEEE Std 830-1998”, The Institute of Electrical and Electronics Engineers, Inc., 1998.

[IID03a] 「ユーザビリティとは何か」、http://www.usability.gr.jp/whatis.html

[IID03b] 「ユーザビリティとは？」、http://www.usability.gr.jp/whatis/whatis001127-1.html

[IPA05]「プロジェクト編成会資料 XX 情報誌リニューアルプロジェクト」、IPA、2005 年.
この資料は独立行政法人情報処理推進機構（IPA）のソフトウェア・エンジニアリング・センター（SEC）にある「事例検索システム」のページ（URL は https://sec.ipa.go.jp/enterprise/index.php?type=s&ccd=n）に、提供社名は「B 社」、資料名は「プロジェクト編成会資料」として登録されており、そこからダウンロードすることができる。（実際にダウンロードするためには、IPA にユーザ登録する必要がある。）

[IPA06a] 「IT スキル標準 v2」、独立行政法人情報処理推進機構 IT スキル標準センター、平成 18 年 4 月 1 日.
なおこの資料は、以下の URL からダウンロード可能である。
http://www.ipa.go.jp/jinzai/itss/download_V2.html

[IPA06b] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター編、「経営者が参画する要求品質の確保～超上流から攻める IT 化の勘どころ～第 2 版」、(株)オーム社、平成 18 年 5 月 25 日、ISBN4-274-50076-4.

[ISO89] 日本工業標準調査会情報部会審議、「開放型システム間相互接続の基本参照モデル－安全保護体系 JIS X 5400-1991 （ISO 7498-2：1989)」、(財) 日本規格協会、平成 3 年 8 月 1 日.

[ISO98a] 日本工業標準調査会情報部会審議、「ソフトウェアライフサイクルプロセス－構成管理 TR X 0018：1999 （ISO／IEC TR 15846：1998)」、(財) 日本規格協会、平成 11 年 10 月 30 日.

[ISO98b] 日本工業標準調査会審議、「人間工学－視覚表示装置を用いるオフィス作業－使用性についての手引き JIS Z 8521:1999 （ISO 9241-11:1998)」、日本規格協会、平成 11 年 3 月 20 日.

[ISO00] 日本工業標準調査会審議、「品質マネジメントシステム－基本及び用語 JIS Q 9000：2000 （ISO 9000：2000)、日本規格協会、平成 12 年 12 月 20 日.

[ISO01] 日本工業標準調査会審議、「ソフトウェア製品の品質－第 1 部：品質モデル JIS X 0129-1：2003 （ISO／IEC 9126：2001)」、日本規格協会、平成 15 年 2 月 20 日.

[JIS03] 日本工業標準調査会審議、「マネジメントシステムのパフォーマンス改善－品質機能展開の指針 JIS Q 9025：2003」、(財) 日本規格協会、平成 15 年 2 月 20 日.

[JUA03] 「情報システムのユーザ満足度プロジェクト報告書」、(社) 日本情報システム・ユーザー協会、2003 年.

[JUA04] 「エンドユーザによるビジネスシステム定義の進め方 平成 15 年度ビジネスシステム定義研究プロジェクト報告書」、(社) 日本情報システム・ユーザー協会、平成 16 年 4 月.

[JUA05] 「システム・リファレンス・マニュアル（SRM)」、(社) 日本情報システム・ユーザー協会、2005 年 9 月.

[JUA06a] 「ユーザー企業ソフトウェアメトリクス調査 2006」、(社) 日本情報システム・ユーザー協会、2006 年.

[JUA06b] 「企業 IT 動向調査 報告書 2006 年版」、(社) 日本情報システム・ユーザー協会、2006 年.

[JUA07a] 「システム・リファレンス・マニュアル 2007 年版」(社) 日本情報システム・ユーザー協会、2007 年.（未刊行）

[JUA07b] 「ユーザビリティ研究会報告書 2007 年版」、(社) 日本情報システム・ユーザー協会、2007 年.（未刊行）

[JUA07c] 「企業 IT 動向調査 報告書 2007 年版」、(社) 日本情報システム・ユーザー協会、2007 年.（未刊行）

[MAR89] ジェームズ・マーチン著、竹林則彦監修、三菱 CC 研究会 IE タスクフォース訳、「インフォメーション・エンジニアリング I 統合化 CASE のための方法論」、(株) トッパン、1991 年.

[MAR90a] ジェームズ・マーチン著、竹林則彦監修、三菱 CC 研究会 IE タスクフォース訳、

「インフォメーション・エンジニアリングⅡ 統合化 CASE による計画と分析」、(株) トッパン、1992 年.

[MAR90b] ジェームズ・マーチン著、竹林則彦監修、三菱 CC 研究会 IE タスクフォース訳、「インフォメーション・エンジニアリングⅢ 統合化 CASE による設計と製作」、(株) トッパン、1994 年.

[MAS03] 増永良文著、「リレーショナルデータベース入門 [新訂版] ーデータモデル・SQL・管理システムー」、Information & Computing – 43、サイエンス社、2003 年.

[MET06a] 「EA ポータル」は、次の URL からダウンロードできる。

http://www.meti.go.jp/policy/it_policy/ea/index.html

[MET06b] 「情報システムの信頼性向上に関するガイドライン」は、次の URL からダウンロードできる。

http://www.meti.go.jp/press/20060615002/guideline.pdf

[NON96] 野中郁次郎、竹内弘高著、「知識創造企業」、東洋経済新報社、1996 年.

[NON03] 野中郁次郎、紺野登著、「知識創造の方法論 ナレッジワーカーの作法」、東洋経済新報社、2003 年.

[POW01] Power, N., "Variety and Quality in Requirements Documentation," Proc. of REFSQ2001, 2001.

[PRE05] ロジャー・S. プレスマン著、西康晴他監訳、古沢聡子他訳、「実践ソフトウェアエンジニアリング ソフトウェアプロフェッショナルのための基本知識」、日科技連、2005 年.

[RYU06] 劉功義、「プロジェクト計画における要求整理方法の提案 プロジェクトマネジメント学会 2005 年度第 2 回研究員会フォーラムでの講演資料より」、PM 学会研究委員会、平成 18 年 3 月 1 日.

[SAN94] J. サンダース、E. カラン著、原田曄他訳、「ソフトウェア品質向上のすすめー新しいソフトウェア開発の標準」、(株) トッパン、1996 年.

[SAT05] 佐藤正美著、「データベース設計論ーT 字型 ER 関係モデルとオブジェクト指向の統合をめざして」、ソフト・リサーチ・センター、2005 年.

[SIM05] 清水吉男著、「[入門+実践]要求を仕様化する技術表現する技術～仕様が書けていますか」、(株) 技術評論社、平成 17 年.

[SOM97] Ian Sommerville、Pete Sawyer 著、富野壽監訳、「要求定義工学プラクティスガイド」、(株) 構造計画研究所、2000 年.

[TAM03] 玉置彰宏著、「ソフトウェア工学の勧め 第 1 章 ソフトウェア工学とは」. この原稿は以下の URL からダウンロードできる。

http://www.rsch.tuis.ac.jp/~tamaki/software_engineering/Chap_01.pdf

[TUB05] 椿正明著、「名人椿正明が教えるデータモデリングの技 データ中心システム開発原論」、翔泳社、2005 年.

付 2. 要求仕様書のプロトタイプ

標　題	内　容
1. はじめに	
1.1. 目的	この要求仕様書の目的、想定される読者
1.2. 範囲	開発するソフトウェアの種類。説明。このソフトウェア開発で達成しようとする目的、ゴール、得られる利益。「業務要求」もここに記載する。
1.3. 専門用語、頭文字語、略語の定義	
1.4. 参照	参照する資料・文献
1.5. 概要	この要求仕様書の構成
2. 全体の記述	
2.1. 製品についての考え方	システムを構成する他の部分との関係、インタフェース
2.2. 製品の機能	３で述べる機能の概要
2.3. ユーザの特性	想定されるユーザの経験や技術レベルなどで特記するべき事項
2.4. 制約	このソフトウェアの開発に伴う制約事項
2.5. 前提	このソフトウェア開発の前提条件
2.6. 先送りされる要求事項	
3. 要求と仕様	要求仕様書の本体部分。「USDM 表記法」で記述する。
付録	
索引	

付 3. 要求仕様書のテンプレート

					優先度	備考欄
カテゴリ名 （記号）	要求	（要求番号）	ここに要求を記述する。			
		理由	要求の背景や理由について記述する			
		説明	要求について必要に応じて説明する。仕様とは見なされない。セルを広げて図を貼り付けることも可能。			
		要求	（要求番号）	ここに階層化された要求を記述する。要求番号は一段下げられる。		
			理由	範囲を狭めた要求について、背景や理由を記述する。		
			説明	要求について必要に応じて説明する。仕様とは見なされない。		
		<<仕様分類名>>		主分割記号・・・全体を通して共通の分割基準として決める。		
			<<仕様分類名>>	補助分割名・・・主分割の中に異なるテーマの仕様が混在する場合、補助分割記号を使って純度の高い集合を作る。		
		□	（仕様番号）	ここに仕様を記述する。		
		□	（仕様番号）			
			<<仕様分類名>>			
		□	（仕様番号）			
		□	（仕様番号）			
		□	（仕様番号）			
		□	（仕様番号）			
			理由	仕様について、必要に応じて背景や理由を記述する。		

付 4. 画面／帳票定義書の記載項目

１．画面／帳票のイメージ
２．入出力データ
３．入出力データ項目
４．ロジック
　① 入力チェック
　② データ加工
　③ DB 更新の処理
　④ 分類
　⑤ その他
５．画面のカスタマイズ
　① 画面表示
　② 画面遷移
６．異例処理
７．パフォーマンスに関する要件

（S 社の資料を基に、一部修正した。）

付 5　「USDM 表記法による要求の仕様化について」

（清水吉男氏のレジメ）

USDM表記法による
要求の仕様化について

JUAS「UVC研究プロジェクト」資料
2006年8月28日

(株)システムクリエイツ
代表取締役 清 水 吉 男
shimz@mb.infoweb.ne.jp
URL=http://homepage3.nifty.com/koha_hp

0. 本日の内容

1. 要求仕様の問題
2. USDMの特徴
3. 要求を表現する
4. 仕様を表現する
5. 画面仕様にもUSDMを適用する
6. 品質要求の扱い
7. USDMの応用

USDM:Universal Specification Description Manner の略
参考文献:(1)、(2)

1. 要求仕様の問題

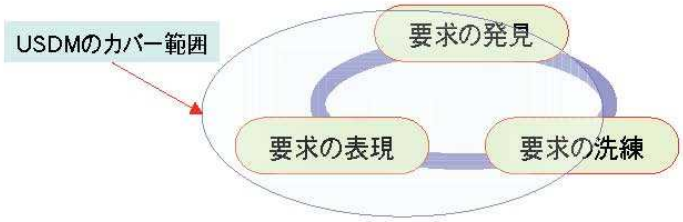
- “仕様書”としてのレベルに達していない状態
  - 仕様が漏れているのに気づかない
  - 仕様間で衝突していても気づかない
  - ここからはまともなテストケースは作れない
- 関係者間で合意ができないままの開始
- 頻発する設計途中での仕様変更
  - そこにある仕様書と乖離が始まる
- 保守性などの品質要求が抜け落ちている
  - 以降の派生開発で崩壊していく

2. USDMの特徴

- USDMによる表記法には、次のような特徴がある
  - 「要求」を表現する
  - 「理由」で要求で補強する
  - ゴールとしての「範囲」を見せる
  - 階層構造で要求と仕様を捉える
  - 制御された要求の中で仕様を抽出する
  - 要求書と仕様書を一体化する
  - 徹底的に「分類と整理」にこだわる

要求仕様の表現にFocus

- 要求開発には3つの要素がある



- せっかく発見・抽出された要求も、適切に表現されなければ仕様が漏れる
  - USDMは「表現」に重点を置き、他の2つをカバーする表記法

「要求仕様書」に対する考え方が違う?

- 「要求仕様書」と「機能仕様書」と何が違うのか?

	要求仕様書	機能仕様書
文書の特徴	「要求」すなわち実現して欲しいこと(Requirement)について、"作ることの関係者"が認識を特定(Specify)できている文書 保守性などの作り方の品質要求が加わる	そこで提供する「機能」(Function)についての具体的な振る舞いなどを客観的なレベルで書かれた文書
関係者	依頼者、設計者、検証者など実現にかかわる人たちで特定されている	誰が見るか分からない

実現を誘導する文書

“要求”を表現する

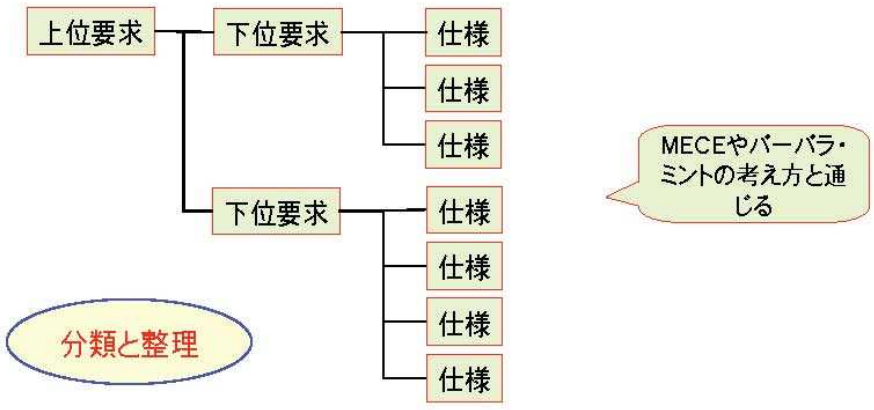
- 一般には、「要求」はどこにも表現されていない
  - ヒアリングの場などで“言葉”で交わされることがあるが・・・
- USDMによる表記法では「要求」を文章で表現する
  - 「要求」とは実現したいことの“ゴール”
    - 機能
    - 品質
    - その他の制約
  - 「要求」は、表現の仕方によって「仕様」を束ねる力がある

要求	MAL01	受信した電子メールをキーワード検索したい
要求	PRT07	インクの残量を調べて知らせて欲しい
要求	BYY03	(ネット販売)購入の確認操作では誤操作が入る余地を排除して欲しい
要求	FLE10	設定データが変更されたときは、あとで再現できるように必要な情報と一緒に保存すること

- 「要求」によっては“粒度”(範囲)に差が生じる

要求と仕様を階層構造で捉える

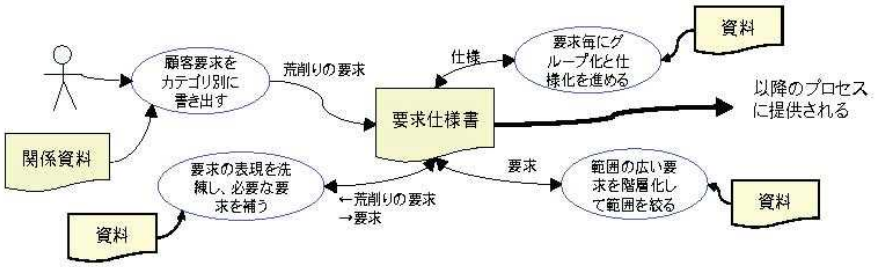
- USDMでは、要求と要求仕様を2種類の階層構造の中で捉える



- 階層構造=仕様がモレない基本的な構造

要求書と仕様書を一体化する

- 従来は、新しい文書を生成する過程で、内容を深めていった
  - 内容が複製された場合、その後の変更で保守しきれないし、
  - 内容が複製されなければ、表現が変化して参照できない状態になる
- 階層で表現することで、「要求書」と「仕様書」を一体化できる
  - いくつかのプロセスで加筆し洗練していく中で「要求書」から「要求仕様書」変化する
  - 機能追加や仕様の変更への対応が容易



3. 要求を表現する

- ここでのポイント
  - 「要求」はゴールであり、「範囲」を見せるように表現する
  - 範囲が広い場合は、階層化して範囲を狭める
  - 「理由」と組み合わせて要求を調整する

「要求」の役割は、有効な「仕様」を引き出すことにある

要求がないと仕様の漏れに気づきにくい

- 従来は機能名の下に、いきなり「仕様」が列記される

> 3.5 座席予約システム
> ・講演一覧から選ばれた講演の座席マップを表示する
> ・画面に表示された座席マップをマウスでクリックすることで予約できる
> ・予約できた座席は、色を茶色に変える
> ・「座席予約完了」ボタンが押されれば精算画面を表示する
> ・精算画面は、それまで予約された座席番号と金額と合計金額を表示する

- この状態（構成）では、仕様の漏れに気づきにくい
  - タッチの差で他の端末から先に予約されるケースはないのか?
  - 「座席予約完了」後にキャンセルは受付けなくても良いのか?

要求は「範囲」を見せるように表現する

- 実現してほしいことの「範囲」を表現する
  - 実現したい「範囲」を表現することで、関係者の認識のズレを矯正する

要求	MAL01	電子メールを検索したい

  - これじゃ、「その人のPCに残されている電子メールを探す」ということしか分からない
  - 「電子メール」と言われても・・・
  - 「探し方」だってどのような方法をイメージすればよいのか分からない
  - 探した後はどうするの?

機能の上位要求には、「入力ー変換ー出力」の動きを持つ

要求	MAL01	事前に指定された受信および送信した電子メールをキーワードで検索してメーラー上で再利用したい

要求には“理由”がある

- 要求には、それが存在する「理由」がある
  - 背後にある「理由」を探ることが、仕様を外さない一つの方法
- 「理由」が、要求の表現を矯正する
  - 顧客の「要求」を正しい位置に誘導する
  - 「理由」によって別の仕様を引き出すこともできる
  - “理由”が考慮されることで設計の品質に差が出る

要求	PRT07	インクの残量を調べて知らせて欲しい

  - という要求の表現では要求の意図が絞り込めないし、実現したいことの「範囲」も見えない

要求	PRT07	インクの残量を調べて、最初の段階で印刷できない可能性があれば印刷依頼者のPC上に知らせて欲しい
	理由	印刷途中で印字がかすれたくないから

この要求は実現できない?

分割・階層化して要求の範囲を狭める

- 要求の「範囲」が広いままだと
  - ① 仕様の抽出そのものがモレてしまう
  - ② 必要な仕様が抽出されていないことに気付かない
- 分割・階層化して要求の範囲を狭める

要求	MAL01	事前に指定された受信および送信した電子メールをキーワードで検索してメーラー上で再利用したい
	理由	メールが多くて、関連するメールを 探せない

  - 上位要求には、シナリオとしていくつかの動きをその範囲に持っている
  - 上記の要求には3～4個の”動詞”が含まれている
- 階層は2層でとめる
  - 要求の粒度を工夫する

要求の分割と階層化

- 階層化した要求にも「理由」を忘れないこと

要求	MAL01	事前に指定された受信および送信した電子メールをキーワードで検索してメーラー上で再利用したい	
	理由	メールが多くて、関連するメールを 探せない	
	要求	MAL01-01	検索対象のメールボックスを指定してグループ化できる
		理由	無関係なメールで振り回されたくない
	要求	MAL01-02	グループに対していくつかのキーワードを組み合わせて検索できる
		理由	可能性のあるキーワードで探したい
	要求	MAL01-03	検索結果を扱いやすく表示し、そこから選択したい
		理由	目的のメールが一つとは限らないので、絞り込めるような操作がしたい
	要求	MAL01-04	選択した電子メールをメーラーに繋いで編集する
		理由	一部を変更するだけで作業の効率化が図れる

スペースの関係で「説明」の欄は省略しています

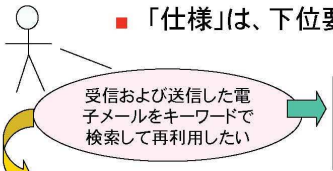
- この後、仕様は、階層化された要求の中で抽出される

4つの分割の基準

- 要求を分割する基準として4つの基準を導入する
  - ① 時系列分割
    - 時間軸に着目して、要求を分割する方法
  - ② 構成分割
    - 「機能」や「構成」で要求を分割する方法
  - ③ 状態分割
    - 「状態」という概念で要求の階層化
  - ④ 共通分割
    - 共通する機能を切りだして独立させる方法
- 読み手も、同じ「分割の基準」を共有することで、要求や仕様の漏れに気付きやすくなる.
  - 私的な基準では混乱するだけ

ユースケースは「要求」に対応する

- 「ユースケース」→「上位要求」
- 「シナリオ」 ——→「下位要求」に対応する
  - 「仕様」は、下位要求の下に展開すれば良い



シナリオ:
1)検索対象のメールボックスを指定してグループ化する
2)いくつかのキーワードを組み合わせて検索する
3)多いときは検索結果から絞り込むための検索をする
4)検索されたメールを選択し内容を表示する

要求	MAL01	受信および送信した電子メールをキーワードで検索して再利用したい	
	理由	メールが多くて,関連するメールを 探せない	
	説明		
	要求	MAL01-01	検索対象のメールボックスを指定してグループ化できる
	理由		無関係なメールで振り回されたくない
	要求	MAL01-02	いくつかのキーワードを組み合わせて検索できる
	理由		可能性のあるキーワードで探したい
	要求	MAL01-03	検索結果から絞り込むための検索ができる
	理由		件数が多いときに目的のメールを探すのに手間取るから
	要求	MAL01-04	検索されたメールを選択し内容を表示する
	理由		目的のメールを絞り込めるような操作がしたい

4. 仕様を表現する

- ここでポイント
  - すべての仕様は、いずれかの要求に属する
  - 仕様から要求を立てることも
  - 必要なら「認定仕様」を取り入れる
  - 範囲が制御された要求にフォーカスすることで仕様の抽出は容易になる
  - 必要に応じて[理由]や[説明]を付ける
  - <グループ>によって「分割と整理」を徹底する
- 「要求」の枠があることで、仕様化の作業を複数の人で分担しても、仕様が発散しない
  - 「あるべき場所」で仕様が表現される
  - 早い段階で「要求」を安定させることが重要

仕様であることの条件

- 関係者の間で"Specify"できているもの
  - 「要求仕様書」とは、実現が求められている要求(Requirement)について、関係者がその意味を特定できる(Specify)状態にまとめられた文書
- 実現可能性(実行可能性)が見える
  - 設計者が、設計や実装作業の中で仕様を問い合わせることがない状態
- 検証可能性が見える
  - 事前に、検証担当者によって検証法法がイメージできることを確認

要するに、あとで仕様について問合せを発しない状態であり、仕様の変更が発生しない状態

[認定仕様]の考え方もある

- 特殊なケースとして「要求」の状態であっても、それ以上仕様化する必要を認めない状態を「認定仕様」として容認する
  - 納期も主要な要求である
  - 下位要求の中で可能

「仕様」として認めたことを示す

要求	MAL01	事前に指定された受信および送信した電子メールをキーワードで検索してメーラー上で再利用したい	
	理由	メールが多くて,関連するメールを 探せない	
□□□	要求	MAL01-01	検索対象のメールボックスを指定してグループ化できる
	理由		無関係なメールで振り回されたくない
	要求	MAL01-02	いくつかのキーワードを組み合わせて検索できる
		理由	可能性のあるキーワードで探したい

- 「MAL01-01」は、この下に仕様を展開しない

制御された要求の中で仕様を引き出す

- 適切に範囲が制御された要求の中で、その要求を満たすための仕様を引き出す
  - 範囲が制御された要求にフォーカスすることで仕様の抽出は容易になる
  - 仕様の漏れにも気づきやすい

要求	MAL01	受信および送信した電子メールをキーワード検索したい	
	理由	メールが多くて,関連するメールを 探せない	
	要求	MAL01-01	検索対象のメールボックスを指定してグループ化できる
		理由	無関係なメールで振り回されたくない
	□□□	MAL01-01-1	メーラーが管理するメールボックスをグループ名で一覧表示する
	□□□	MAL01-01-2	グループ名が指定されていない場合はグループ名の欄は「空」で表示する
	□□□	MAL01-01-3	新たにグループ名が指定されたときは、グループ名一覧に追加する
	□□□	MAL01-01-4	メールボックスの一覧からメールボックスを選択したグループに指定する
	要求	MAL01-02	いくつかのキーワードを組み合わせて検索できる
		理由	可能性のあるキーワードで探したい
	□□□	MAL01-02-1	検索したいキーワードを入力できる
	□□□	MAL01-02-2	複数のキーワードを「AND」と「OR」で繋ぐことができる
	□□□	MAL01-02-3	キーワードは最大8個まで指定できる

仕様をグループ化して整理する

- 一つの要求の中で抽出される仕様が多くなった時は、それぞれの仕様が扱う“対象”ごとに＜グループ＞にまとめる
  - 要求の「分割基準」を準用する

時系列に整理している

要求	RSV01-03	予約操作を終了したときは、予約を確定させて登録されているメールアドレスに予約状況を送信する
	理由	顧客に予約が成立したことを示すことと、あとで確認できるようにするため
	＜メールアドレスの確認＞	
□□□	RSV01-03-1	予約結果を送信するメールアドレスを確認する為に、登録されているアドレスを表示する
□□□	RSV01-03-2	メールアドレスの変更がある時は、変更を受付ける
□□□	RSV01-03-3	メールアドレス変更された時は、再度入力を求めて確認する
	＜予約番号の付与と予約結果の表示＞	
□□□	RSV01-03-7	今回の一連の予約操作に対して、「予約番号」を付与する
□□□	RSV01-03-8	「ご予約状況」として以下の情報を表示する
	＜メールの送信＞	
□□□	RSV01-03-10	「ご予約状況」と同じ内容の情報を、電子メールで送信する
□□□	RSV01-03-11	送信エラーの状態であれば、画面の印刷を促すメッセージを出す
	＜クレジット処理＞	
□□□	RSV01-03-15	事前に登録しているクレジットカードの番号を使って、合計金額をクレジット会社に送信する

＜グループ＞を先にイメージすることもできる

- 要求を表現している中で、その下で展開される仕様の＜グループ＞が見えることがある
  - 分割基準を遵守すること

時系列に整理している

要求	MAL01-03	検索結果を扱いやすく表示し、そこから選択したい
	理由	目的のメールが一つとは限らないので、絞り込めるような操作がしたい
	＜検索結果の表示＞	
	＜検索メールの表示＞	
	＜メールの中身の表示＞	
	＜メールの選別＞	

- この後の仕様化作業の効率が格段に向上する
  - 仕様化作業を手分けすることも容易になる

仕様から要求を立てることも必要

- 「醜いアヒル現象」
  - 要求が範囲を持つことで、要求と理由を共有しない仕様が見える

要求	MAL01-02	いくつかのキーワードを組み合わせて検索できる
	理由	可能性のあるキーワードを簡単に探したい
□□□	MAL01-02-1	検索したいキーワードを入力できる
□□□	MAL01-02-2	複数のキーワードを「AND」と「OR」で繋ぐことができる
□□□	MAL01-02-3	キーワードは最大8個まで指定できる
□□□	MAL01-02-4	検索されたメールの「Subject」を一覧で見せる

この中に、要求と理由を共有しない仕様が混じっている！

- 対応策
  1. 既存の要求の中から適当な要求を探す
  2. この仕様を扱うために新たに要求を設定する
  3. 要求の範囲を広げて、この仕様を含むようにする

その他の表現上の注意

- 要求や仕様には特定できる「記番号」を付ける
- ペースト作文に制限をつける
- 否定表現（肯定表現）に注意する
- 曖昧な表現を避ける
- 複数の仕様を含む表現を避ける

セルの枠と文章との”隙間”を確保したり、図や表を取り入れて見やすくする工夫が必要

仕様変更率 ≦ 5% を目指す

- 仕様としての表現が適切だったかどうかを測る方法は?
  - 適切な期間で書けた.
  - 合意後の変更が少ない.
  - 仕様関係のバグが少ない.
  - 設計や実装作業に支障を来さない.
- ベースライン設定後の仕様の変更が5%以内を目指す.

$$\frac{\text{変更仕様件数}}{\text{総仕様件数}} \times 100$$

- 5%を越えるようだと、CMMの要件管理が破綻する危険がある
- 仕様書の更新が容易
  - 十分に仕様化された中では、変更される仕様の収容先が簡単に見つかる.

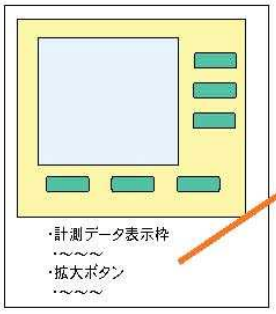
5. 画面仕様にもUSDMを適用する

- ここでのポイント
  - 最近では設定や動作などの多くの操作が「画面」を通じて実現できる
  - 表示や入力ボックスやボタン類に対して、具体的に仕様化する
- 要求仕様の関連文書を体系化しておく
  - 「要求仕様書」を総称として使用し、目的別に用意することは可能
    - 「画面操作仕様書」
    - 「主要機能要求仕様書」
    - 「通信仕様書」など

"要求仕様書"という文字が付いていなくても、要求仕様書としての条件を満たしていれば良い

"仕様書"になっていない

- 表示や操作方法についての主要な仕様が書かれているだけで、必要十分な「仕様」は書かれていない



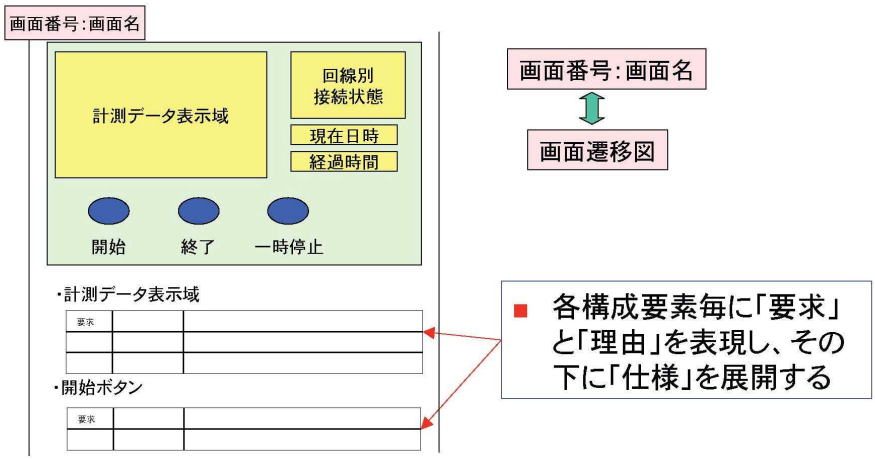
- 計測データ表示枠
  - ここには、接続機器から送られてくる計測データを表示します.
  - 表示枠の右上に、1分間の平均値を少数以下第2位まで表示します.
- 拡大ボタン
  - このボタンを押すと、表示域のゲージが5倍に拡大します.
  - 拡大中に押されると、元のサイズに戻ります.

- 画面レイアウトの下に、簡単な仕様が書かれているだけ
- ボタンや表示枠に対する「要求」は書かれていない

「GUI構築ツール」で解決するか?

画面仕様にもUSDMの表記法を活用する

- 個々の画面の中のそれぞれの構成要素ごとに「要求」を表現する
  - 通常の要求仕様の表記法をそのまま適用できる



個々の画面要素ごとに要求と仕様を表現する

- 個々の画面レイアウトに含まれるすべての「要素」に対して「要求」を表現し、その下に「仕様」を展開する
  - 一般に、要求の階層は浅い

GUIツールで確認する項目

				GUI
計測データ表示域	要求	DS.A.01	計測したデータをリアルタイムに表示する	
		理由	接続先の状況に素早く対応する必要があるため	
		<初期表示>		
	■■■	DS.A.1.1	計測を開始するまでは、画面の中央に"NONE"と表示する	
		<通常表示>		
	■■■	DS.A.1.4	計測したデータは、1レコードを1行に表示する	●
	■■■	DS.A.1.5	項目の境界に1桁のスペースをに挿入する	
	■■■	DS.A.1.6	30行を越えた時点で右側にスクロールバーを表示する	●
		<平均値の計算>		
	■■■	DS.A.1.10	計測データを20件取得した時点で、項目#3～7の項目の平均値を算出する	●
	■■■	DS.A.1.11	平均値データは、その後10分おきに20件のデータを加算して算出仕直す	●
		<警告表示>		
	■■■	DS.A.1.15	項目#3～7の項目で、それまでの平均値と10%以上の差が生じたときは赤字で表示する	
	■■■	DS.A.1.16	警告表示は、平均値が算出されてから実施され、その後継続される	●

<グループ>でまとめることで仕様の発散を防いでいる

<グループ>をパターン化しやすい

- ボタンなどの画面要素の要求は、<グループ>を“パターン化”できることが多い
  - <グループ>の一部は、当該ボタン独自の仕様を扱うことになるが、そのほかの<グループ>はほとんど共通になる

    - <このボタンが押せる条件>
    - <押した時のファイル操作>
    - <計測を停止させる方法> ・・・・・・・・・・当該ボタン特有のもの
    - <計測が停止したことの確認方法>・・・当該ボタン特有のもの
    - <次の画面への切り替え>

  - <グループ>のパターン化によって、仕様の抽出が捗る

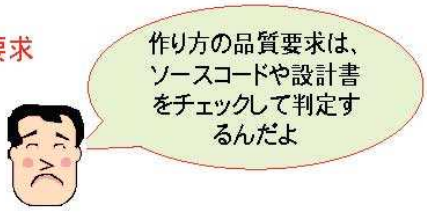
6. 品質要求の扱い

- ここでのポイント
  - 品質も「要求」と「仕様」の形で表現する
  - 「仕様」であることの条件は、通常の仕様と同じ
  - ただし、保守性などの品質特性については、プログラムを実行して確認することはできない
  - 品質に関する要求仕様には、そこに示された仕様を実現すれば、この品質要求は満たしたと「看做す」という基準で持ち込まれることがある

品質特性の分類

- 品質要求には
  - 個々の機能に付随する品質要求
    - パフォーマンス
    - 障害許容性
    - 安全性 など
  - 製品／システム全体にかかる品質要求
    - 操作性
    - 資源効率性
  - 作り方に関する品質要求
    - 保守性
    - 移植性 など

ISO/IEC 9126
JIS X 0129-1



品質要求を仕様化すること

- 品質も具体的に仕様として定義されないものは実現しない
  - 必要な品質特性に応じて達成すべき要求と
  - それが達成したといえるための仕様を記述する

「2秒」は目標

要求	DSP01-02	(応答性)「表示」ボタンを押してから2秒以内に画面の表示が完了すること
	理由	ここで表示するデータの処理量が多く、表示が遅いとハングアップしたと勘違いする可能性があるため
□ □ □	DSP01-02-1	画像処理するデータ量が「2MB」以内の時は、通常通りに処理する
□ □ □	DSP01-02-2	画像処理するデータ量が「2MB」を越えるときは、処理中は「プログレスバー」を表示する

- 上記の仕様を満たせば「2秒以内に表示が完了した」と看做す
  - その時点で活用できる技術などの事情による制限
  - 製品などの設計上の“リソース”の制限 など

保守性も仕様化しないと実現しない

- 保守性や移植性の品質要求も、仕様化しないと実現しない
  - ただし、“品質を織り込む”設計技術が必要
  - プログラムを実行して確認されるものではないが
  - 「実現可能」「検証可能」という条件を満たすこと

要求	QUA.02	**(保守性)**ソフトウェアの保守性を向上させて欲しい.
	理由	この後5年以上、ここから派生製品を展開したいから
□□□	QUA.02-1	一つのクラスに含まれるメソッドは 10個以下とし、それを越える場合は事前に了解をとる(*)
□□□	QUA.02-2	モジュールの複雑度は20以下を原則とし、それを越える場合は事前に了解をとる(*)
□□□	QUA.02-3	モジュールの凝集度も“手順的凝集度”以下になる場合は事前に検討する(*)
□□□	QUA.02-4	処理と管理は明確に分離し、関数の呼び出しの深さが5を越えるときは事前に了解をとる(*)
□□□	QUA.02-5	タスク間でアクセスし合うグルーバル・データを作らない.アクセス時間等による制限から、グローバル・データとなることが避けられない時は、品質検討会議で承認されること. (*) 【説明】割り込み処理との間でのみ共有するケースはこの制限を受けない.

(*)=PLが対応したり、品質検討会議のような組織で対応する

7. USDMの応用

- この表現方法は広い応用範囲を持つ
  - リスク管理に応用
    - リスク(=避けたいこと)・・・要求
    - リスクに繋がる要因・・・理由
    - 回避措置、軽減措置・・・仕様
  - 一般的な問題解決策
    - 解決したいゴール・・・要求、理由
    - その中で具体策としてのアクション・・・仕様

仕様=実行可能であること

参考文献

- 単行本
  - (1)「要求を仕様化する技術・表現する技術」(技術評論社)
- 雑誌
  - (2)「要求の仕様化入門」(Software People Vol.4:技術評論社)
  - (3)「失敗しない派生開発」(Software People Vol.8:技術評論社)

「要求仕様定義ガイドライン」～ＪＵＡＳ・ＵＶＣ報告書～

発行日：平成 19 年 3 月

発行所：社団法人　日本情報システム・ユーザー協会

〒103-0001　東京都中央区日本橋堀留町 1-10-11　井門堀留ビル４階

TEL 03-3249-4102　FAX 03-5645-8493

URL　http://www.juas.or.jp/

本調査は、経済産業省から委託を受け、実施機関として、
社団法人日本情報システム・ユーザー協会（JUAS）が調査研究を実施いたしました。