第二十一年度
ソフトウェア・メインテナンス研究会
報告書

2012年9月

ソフトウェア・メインテナンス研究会
http://www.serc-j.jp/

持続可能性

今年の ICSE in チューリッヒ の基調テーマに「持続可能な世界のための持続可能なソフトウェア」が選ばれたように，今世紀に入ってから「持続可能性」という概念があちこちで取り上げられるようになってきた．そのことは， 現代の科学技術文明がひとつの曲がり角に来ていることの証しであろう．日本でも昨年春のフクシマ原発事故以来，この国の持続可能性について多くの人びとが真剣に考えはじめているようである．

ところで，ソフトウェア・システムの保守は，ハードウェアと同じように考えることはできない．ハードウェアの保守は，そのプロダクトを工場出荷時と同じ状態に戻すことによって持続可能性を維持するのであるが，現実世界のアプリケーションに組み込まれたかたちのソフトウェアは，絶えず変化するアプリケーション環境に対応して常に進化して行かなければその持続可能性を保つことができないという性格をそなえている．ハードウェアの小型化・高速化およびネットワーク技術の普及によって，コンピュータが社会システムのあらゆる部分に組み入れられている現在，ソフトウェアの保守すなわち進化をどのように扱うべきかという問題の重要性はますます高まってきた．

われわれの研究会の活動は，そうしたソフトウェア進化問題の解決に， ささやかではあるが，いささかの貢献を成すことを意図してきた．研究員各位の努力に敬意を表するとともに，さらに多くの同志が当研究会に参加されることを願うものである．

2012年秋

ソフトウェア・メインテナンス研究会
代表幹事　　岸田孝一

目次

第二十一年度研究員名簿 ……4
第二十一年度活動報告……5

1. フォーラム資料
資料1－1　2012 年 3 月 8 日　10 年後の保守を考える……7
資料1－2　2012 年 5 月 18 日　大規模障害に学ぶソフトウェア保守……14
資料1－3　2012 年 7 月 25 日　10 年後の保守を考える……45

2. 作業グループ報告書
MORE REALLY, MORE USEFULLY!～ソフトウエア保守ノウハウの改編～ …… 68
保守作業改善の基盤技術調査……115
10年後のソフトウェア保守を考える ……166
SERCの考える保守とは……215

3. まとめ
幹事よりひとこと……255

4. 次年度募集内容 ……258

第二十一年度研究員名簿

組織名	氏名
(株)NTT データ CCS	馬場 辰男
(株)SRA	古石 ゆみ 石川 雅彦 方 学芬 岸田 孝一
(株)アイ・ティ・フロンティア	田中 創
(株)精密形状処理研究所	長谷川 亨
(株)バイトルヒクマ	信国 智子 三村 千恵子 弘中 茂樹
(株)中電シーティーアイ	諸岡 隆司
(財)経済調査会	押野 智樹
NARA コンサルティング	奈良 隆正
アイエックス・ナレッジ(株)	井瀬 英晶 岡田 浩 田中 一夫
システム企画研修(株)	上野 則男
東芝ソリューション(株)	沼田 恵助 川上 康史 佐井 由美子 野口 大輔 増井 和也
(株)日立ソリューションズ	木部 俊之 鈴木 勝彦 高橋 宏志 高橋 芳広 松本 道春
(有)ウィルビーネットワーク	松尾 好博
個人研究員	伊藤 順一 江尻 武志 大島 道夫 玄間 稔 小林 允 塩谷 和範 中山 優紀 福島 茂雄 丸山 陽一 峰村 圭介

第二十一年度活動報告

ソフトウェア・メインテナンス・シンポジウム

テーマ:ソフトウェア保守の新たな道
日時:2011 年 10 月 14 日(金)
基調講演:「ソフトウェアリポジトリマイニングの研究動向」
京都工芸繊維大学　水野　修　准教授

定例活動

「2. 研究グループ報告書」参照

三島集中研修

日時:2011年12月9日(金)～ 12月10日(土)
場所:三島市　三島商工会議所
内容:
＜12月9日(金)＞
13:30-13:35 代表幹事挨拶
13:35-15:00 基調講演「エスノグラフィを応用したソフトウェア開発の信頼性向上」
平田　貞代　富士通(株)
15:00-15:20 休憩
15:20-17:30 各グループでの検討
18:00-20:30 情報交換会　なごみ亭

＜12月10日(土)＞
9:00-11:00 グループ別討議
桐生市市民文化会館(第2会議研修室)
11:00-12:00 グループ別発表
12:30-14:00 昼食
14:00　解散

SERC　Forum

2012年3月8日(木)18:30-20:00　10年後の保守を考える

2012 年 5 月 18 日(金)13:30-17:00　大規模障害に学ぶソフトウェア保守

2012 年 7 月 25 日(水)13:30-17:00　10 年後の保守を考える

協賛

派生開発カンファレンス2012　http://www.xddp.jp/conference2012.shtml
ソフトウェア・シンポジウム2012　http://sea.jp/ss2012/

１．講　演　資　料

資料１－１

「10 年後の保守を考える」

2012 年 3 月 8 日（水）

■プログラム

18：15～　　受付
18：30～18：35　オープニング
18：35～18：50　ゲストスピーチ
「10 年後の保守を考える」 高橋　芳広氏（株式会社日立ソリューションズ）
18：50～19：00　C グループからの提言
「10 年後の保守を考える」 伊藤　順一氏（中央コンピュータ株式会社）
19：05～19：50　ディスカッション
19：50～20：00　クロージング

■会場：株式会社アイ・ティ・フロンティア神谷町オフィス
東京都港区芝公園四丁目 1 番 4 号
地下鉄日比谷線「神谷町」駅「1 番出口」から徒歩 6 分
地下鉄大江戸線「赤羽橋」駅「中之橋口」から徒歩 9 分
http://www.itfrontier.co.jp/corporate/profile/map.html
http://g.co/maps/35672

ソフトウエアメインテナンス研究会

第２１年度Ｃグループ

「１０年後の保守を考える」

第1回フォーラム

２０１２年3月8日　１８：３０～２０：００

アイ・ティ・フロンティア

本日のアジェンダ

＜フォーラム＞

18:15～	受付
18:30～18:35	オープニング
18:30～18:50	ゲストスピーチ 「10年後の保守を考える」 高橋　芳宏氏（株式会社日立ソリューションズ）
18:50～19:00	Cグループからの提言 「10年後の保守を考える」 伊藤　順一氏（中央コンピュータ株式会社）
19:05～19:50	ディスカッション
19:50～20:00	クロージング

＜フォーラム終了後＞

20:00～	移動
20:15～	ゲストを囲んで懇親会（有志）

なぜ１０年後の保守を考えるのか

産業能率大学が６０歳以上の会社員を対象に実施した仕事に対する意識調査で以下の結果が公表されている。

自分の技能や知識を社内で伝承できていない（回答者の３０%）

なぜ伝承できないのか

・伝承する相手がいなかった。・・・４４．４%

・伝承を求められなかった。・・・３７．４%

伝承するためには…

・シニア層のノウハウを伝承する仕組みを考える必要がある。

→保守の断捨離とは（ノウハウの棚卸と整理）

・保守に求められる資質とは…

２１年度Ｃグループの研究方針

・運営方法（前半）

フォーラム形式を主体とした活動とする

・開催場所は、アイ・ティ・フロンティア本社で開催

・開催時間は１８時開催とし、会前半でテーマ発表

・後半でテーマに沿った討議を実施

＊成果の公開方法については別途議論

・運営方法（後半）

・フォーラム結果のまとめ

今年度Ｃグループ参加メンバー

㈱ｱｲ･ﾃｨ･ﾌﾛﾝﾃｨｱ	田中　創
㈱ｱｲ･ﾃｨ･ﾌﾛﾝﾃｨｱ	玄間　稔
㈱ｱｲ･ﾃｨ･ﾌﾛﾝﾃｨｱ	丸山　陽一（リーダー）
中央ｺﾝﾋﾟｭｰﾀｰ㈱	伊藤　順一
ｱｲｴｯｸｽ・ﾅﾚｯｼﾞ㈱	井瀬　英晶
ｱｲｴｯｸｽ・ﾅﾚｯｼﾞ㈱	岡田　浩
東芝ｿﾘｭｰｼｮﾝ㈱	佐井　由美子

ソフトウェア・メインテナンス研究会　Cグループフォーラム

「10年後の保守を考える」

主催：ソフトウェア・メインテナンス研究会（Cグループ）

保守の現場では、保守のノウハウが伝承されておらず、これからの保守を考えた時、無人化の問題が保守の現場の生産性や品質に大きな影響を与えるのではと杞憂しています。日本企業で基幹系システム（大型汎用機やオフコンによるレガシーシステム）を最初に構築し、これまで運用・保守を行ってきたベテラン・エンジニアがそろって定年を迎え、今後の企業システムのメンテナンスが困難になるといわれる問題がありました。

ITの世界では、システム開発の主流は汎用機からオープン系へ移行しているため、若手エンジニアで汎用機やCOBOLの知識を持っている人は少なくなっています。
システム開発運用の現場では、ベテラン・エンジニアがレガシーシステムを担当し、若手がクライアント／サーバシステムやWebシステムを取り扱うケースが多いため、技術的にも業務知識的にもノウハウの継承ができていないことがほとんどです。

このため、企業にとっては今後もレガシーシステムを使い続けるのはリスクとなる可能性がありますが、単純に汎用機をオープン系に置き換えるだけのプロジェクトは、業務に対するインパクトがないため積極投資しづらいというジレンマがあります。
これらの課題に対して、これから10年後の保守を考えた時、誰が誰に何を伝承すべきなのか考える必要があると考えています。
この点をテーマとし、Dグループの高橋氏をゲストに迎え、Cグループのフォーラムを開催しました。

１０年後の
ソフトウェア保守を考える

～１０年後の経験より～

2012年 3月　8日
SERC　Dグループ　高橋　芳広
Mail:yoshi-tk@acm.org

それは１０年後に起こった

背景

- あまり売れ無かった製品
  →保守料では体制が維持できない
- 初期不良が無くなった後、最近１０年間は問題なし
  →経験者が育たない
- それでも問題がおきれば対応しなければならない

発生した課題とその結末

- **誰がやるの**
  →製品知識は誰もない
  →ある程度スキルがあれば誰がやっても同じ
- **ソースは何処**
  →バックアップの媒体から復元
  →言語は経験者なら容易
  （細かい部分は文法書）

発生した課題とその結末（１）

- **調査資料は**
  →過去のファイルを引っかき回して発見（仕様書等、ISO9000準拠）
  →特に以前のトラブルの調査資料・報告書が役に立った
- **再現環境は**
  →システムバックアップ有り
  →しかし、再構築が困難
  特にMFの部分

発生した課題とその結末（２）

- **コンパイル環境は**
  →前提OS、コンパイラ等のVRリスト媒体作成用ツール
  →揃えるのが困難
  →システムバックアップが必要
- **直しても大丈夫か**
  →周囲の影響は文字列検索
  →運用への影響が不安
  （バグもまた仕様）

発生した課題とその結末（３）

- **そして問題の結末は**
  →過去の障害事例を元に、回避策を提示
  →その後再発せず。落ち着いた

１０年後のために（教訓めいたもの）

- **１０年後いきなり障害対応は困難**
  →１０年後にできるのは是正保守のみ（それに向けた対応）
  →改良が必要なら、１、２年に一度、小規模な改良を継続による技術の継承
- **過去の保守資料（障害対応メモ、報告書）**
  →出来れば紙の方が参照が早い
- **コンパイル・リンク環境はシステムバックアップが必要**
- **可能な限り運用回避を目指す**

- SERC　Dグループは

１０年後を見据えて、保守のノウハウ蓄積のための書籍の執筆に取り組んでいます

一緒に参加しませんか

ソフトウエアメインテナンス研究会

「私が考え中の１０年後の保守」

２０１２年３月８日

ＳＥＲＣ　Ｃグループ　伊藤　順一

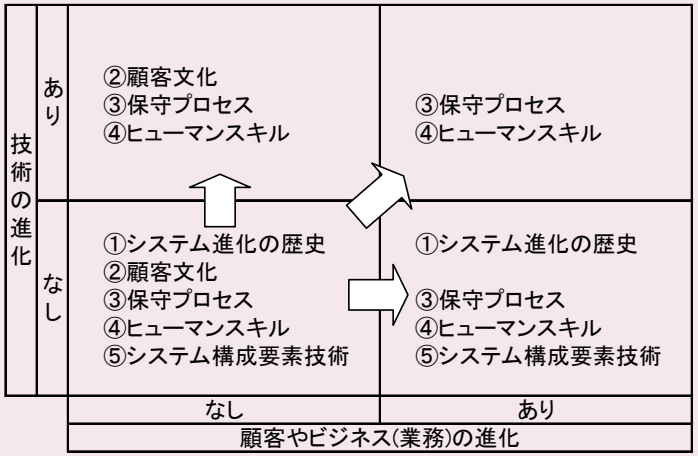
10年後の保守は何が変わっているか？

- → 10年後もあまり変わっていないのでは？保守プロセスは変わらない。技術要素や言葉が変わるだけ。
- → ソフトウェアの大衆化(携帯用アプリ)・クラウド化・ビックデータ化で、保守プロセスも多様化する？
- → ネットワークセキュリティの発達で、自宅で保守できるようになる？
  (大地震発生時帰宅難民にならずにすむため、そうなって欲しい。国をあげて取り組むべき課題だ！)
- → セキュリティ及びリスク管理のさらなる強化で、ソフトウェア保守がしにくくなっている？
- → 海外の人と共同で保守するようになる？(損保も生き残りのために海外に進出している)

誰が誰に何を伝承すべきなのか考える

- → ソフトウェア保守技術？の捨てる部分と継承すべきことを見極める必要あり？(断捨離)
- → 言語も含め、システム構成要素技術は継承の必要がないのでは？
- 前提：ソフトウェア保守者は自ら開発は行なわず、開発支援とソフトウェアの受入を行なう。
- ①システム進化の歴史：どのようにしてシステムを拡張させてきたか
- ②顧客文化　　：業務、用語、仕事の進め方、システム運用
- ③保守プロセス　　：顧客やシステム形態にあったソフトウェア保守手順のルール化と運用
- ④ヒューマンスキル　：対人関係能力、文章技術、物事の考え方、精神
- ⑤システム構成要素技術：ＯＳ、ミドルウェア、言語、DBMS、オンライン・バッチ処理方式、システム間インターフェイス、フレームワーク

誰が誰に何を伝承すべきなのか考える



資料１－２

大規模障害に学ぶソフトウェア保守

～ソフトウェア保守者が見た みずほ銀行オンライン障害～

2012 年 5 月 18 日(金)

■プログラム

13:00 - 13:30 受付
13:30 - 13:35 開会の挨拶
13:35 - 13:55 みずほ銀行障害の概要
　SERC 研究員 高橋 宏志
13:55 - 14:15 システム障害調査委員会の報告書 概説
　SERC 研究員 増井 和也
14:15 - 15:00 SERC の提言１ 組織論，マネジメントの観点から
　SERC 研究員 弘中 茂樹
　SERC 研究員 大島 道夫
15:00 - 15:30 SERC の提言２ ソフトウェア保守技術の観点から
　SERC 研究員 高橋 芳広
15:30 - 15:40 休憩
15:40 - 16:40 質疑応答
16:40 - 16:50 クロージング

■会場：八丁堀区民館　６号室
東京都中央区八丁堀四丁目 13 番 12 号
中央区コミュニティバス(江戸バス)北循環 3 番　八丁堀駅バス停　徒歩 5 分
東京メトロ日比谷線または JR 京葉線八丁堀駅下車 A2 番出口　徒歩 2 分

なお、参考資料の「みずほ銀行　障害報告書」は、以下にあります。
http://www.mizuhobank.co.jp/company/release/2011/pdf/news110520_4.pdf

SERCフォーラム　セッション1

みずほ銀行障害の概要

2012年5月18日

SERC研究員　高橋　宏志

本資料は、SERCフォーラム「大規模障害に学ぶソフトウェア保守～ソフトウェア保守者が見た　みずほ銀行オンライン障害～」の
冒頭で、後のセッションに先立って情報共有するために作成したものです。
みずほ銀行のホームページに掲載されている「調査報告書」（2011.5.20　システム障害特別調査委員会）をもとに、起こった事実を
時系列に整理したものであり、解釈誤りなどがありましたら、作成者の責に帰するものです。

月日	時	対外影響	トラブル	フォロー	連携
3/14(月)	9	＜障害の影響＞ ①為替処理の遅延 ②営業店事務の開始遅延及び取引停止 ③ATMの利用停止及び利用制限 ④ダイレクト・チャネルの利用制限 ⑤その他	10:16 義援金口座a への大量振込でオンライン照会不可	11:30 義援金口座a'（リーフロ） 新規開設	＜凡例＞ hh:mm xxxxx 日時が明示された事象 xxxxxxxxxxxxx 日時が明示されない事象 xxxxxxxxx トラブルの原因 xxxxxxxxxxxxx 事象のグルーピング
	10				
	11				
	12				
	13		A社義援金大量データ	リミット値とは、処理効率や能力等のリスースの範囲内で問題なく処理できる限界値のこと	
	14				
	15				
	16				
	17				
	18			1口座あたりの処理可能な件数のリミット値を上回った。夜間バッチのリミット値は、本支店を40のブロックに分けてそれぞれ設定されていた。	
	19				
	20				
	21				
	22		22:07 振込夜間バッチ異常終了（義援金口座aがリミット値を上回った）	3/14分夜間バッチ	
	23		リミット値を拡大し再実行しようとしたが処理結果データの一部欠落		
3/15(火)	0				
	1			TARGETによって30,000JOB/日の夜間バッチが自動実行されていた	03:30 ITシステム統括部が
	2				
	3			データ復元作業の長期化（8H）	05:00 担当役員が営業店開始時間
	4				
	5			営業店開局特別処理	
	6			07:00 夜間バッチ中断	07:00 担当役員が頭取、副頭取に営業店端末開局を優先して対策中であることを報告
	7		為替未送信データ発生	DJS切替	
	8			取引抑止作業（100業務）（2.5H）	
	9	営業店事務の開始遅延（1時間25分）			09:00 事務サービス推進部長がBCP発動
	10	融資、ローン、外為等を取引抑止	10:25 営業店端末開局		
	11			為替個別打鍵（OK）	
	12			リリースオペ（NG）	12:30 事務サービス推進部長がリリースオペ指示
	13			リリースオペの前提処理が全銀締切時間に間に合わなかった	
	14				「代り金引落未済の振込データの送信」をリリースオペという。
	15			15:30 全銀送信締め	
	16				
	17	17:00 3/15分未送信為替（約31万件）			
	18				
	19				
	20				
	21	21:45 3/15分未送信為替（約38万件）	B社義援金大量データ	3/14分夜間バッチ（継続）～3/15分夜間バッチ	21:45 担当役員から頭取に「未送信為替（約38万件）」を報告
	22				22:00 経営陣が障害対策TFを設置
	23				
3/16(水)	0				
	1				
	2				
	3				
	4				
	5				
	6				06:00 担当役員から経営陣に重要夜間バッチの優先(営業店端末開局遅延の可能性)を報告。経営陣は、11:00までに開局することを指示。
	7		07:17 振込夜間バッチ異常終了（義援金口座bがリミット値を上回った）	営業店開局特別処理	
	8	08:00～08:33 ATM停止	ATM参照日付変更もれ	夜間バッチ中断	
	9	営業店事務の開始遅延（2時間12分）		DJS切替	
	10			参照日付変更＆ATM再起動	
	11	融資、ローン、外為等を取引抑止	11:12 営業店端末開局	取引抑止作業	
	12				
	13				
	14		為替未送信データ発生		
	15	3/16 15:00～3/17 09:00 ATM振込予約停止			
	16				
	17	3/16分未送信為替			
	18	3/16分被仕向為替未処理			
	19				
	20				
	21	一部の時間帯で、ダイレクトチャネル（みずほダイレクト・e-ビジネスサイト・法人ＥＢ）の利用制限		19:20～4:13 異常終了MSGを誤認し義援金口座aのリミット値拡大＆振込夜間バッチ再実行（4回繰返し） 異常終了解消せず	21:00 担当役員から障害対策TFに未送信為替対策が難航していることを報告。頭取から以下の指示。 ・3/17の営業店端末開局は早めにする ・週内の対策が出来ない場合は3連休で対策する
	22		オンライン日替り処理もれ	3/15分夜間バッチ（継続）	
	23				

月日	時	対外影響	トラブル	フォロー	連携
		3/16 15:00 ～ 3/17 09:00	オンライン日替り処理もれ	19:20～4:13 異常終了MSGを誤認し 義援金口座aのリミット値拡大 & 振込夜間バッチ再実行 （4回繰返し） 異常終了 解消せず 3/15分 夜間 バッチ （継続）	
3/17(木)	0	00:00～05:20 ATM停止			
	1				
	2				
	3				
	4				
	5	ATM振込 予約停止		義援金講座bのデータを除外し再実行 05:20 義援金口座bのデータ以外の 振込夜間バッチ完了	
	6			営業店開局特別処理 05:30 夜間バッチ中断	
	7		為替未送信 データ発生	DJS切替	
	8			取引抑止作業	
	9	営業店事務の開始遅延 （1時間46分）			
	10			10:46 営業店開局	
	11	融資、ローン、外為等を 取引抑止			
	12				
	13		取引履歴 退避漏れ	13:30 不要データ削除手動実行	
	14				
	15				
	16				
	17	3/17分 未送信為替	17:20 STEPS 異常終了		
	18	17:20～21:36 ATM停止			
	19	3/17分 被仕向為替未処理			
	20				
	21				
	22		22:46 必要データの喪失判明	3/15分 夜間 バッチ （継続）	
	23	3/17 15:00 ～ 3/18 09:00 ATM振込 予約停止		喪失データ の特定 （5H）	23:00 障害対策TFで協議。 頭取からATM等の利用制限を 指示(3/18～21)。
3/18(金)	0				
	1				
	2				
	3				
	4				
	5				
	6			3/18は、営業店開局特別処理及び営業店端末開局遅延に 関する記載はない。	
	7				
	8				
	9				
	10	10:00過ぎ 未送信為替の送信開始			
	11				
	12				
	13				13:30 障害対策TFのオペレーションルーム を設営することを周知
	14				
	15				
	16				
	17	3/18分為替未送信			
	18	3/18分被仕向為替未処理 （3/16～18：計101万件）			
	19			喪失データ 再作成 （11H）	3/18 19:00～3/21 20:00 断続的に障害対策TFを開催(7回)。 頭取からATM等の利用制限継続 を指示(～3/22)。
	20			3/15分 夜間 バッチ （継続）	
	21				
	22				
	23	3/18 19:00 ～ 3/22 08:00 店舗内 ATM 利用 停止			
3/19(土)				19:05 3/15分夜間バッチ終了	
3/20(日)				夜間バッチの自動化	
3/21(祝)				3/16～21分 夜間 バッチ	
3/22(火)		3/18分まで為替未送信解消 3/22分為替未送信（16万件）		日中オンラインと夜間バッチの並行処理	障害対策TFがオペレーションWGを設置
3/23(水)		3/23分為替未送信(1千件) （3/16～23：計120万件）		3/22分夜間バッチから通常の自動運行に切替	
3/24(木)		為替未送信解消			

3/18
00:00
～
3/23
07:00

店舗外
ATM
利用
停止

一部の時間帯で、ダイレクトチャネル（みずほダイレクト・e-ビジネスサイト・法人ＥＢ）の利用制限

MHBK システム障害特別調査委員会 報告書 概説

2012年5月18日

ソフトウェア・メインテナンス研究会（SERC）幹事
増井 和也

ソフト開発に保守は含まれるのか？

発表者の自己紹介に代えて

- 貴方は作る人かも？ でも，私は直す人
- これまで稼動後直す必要の無い（有価値）ソフトウェアに出会ったことはない
- 私は，作ること（開発）だけが注目されるこの産業は未成熟だと感じている。
- MHBKシステム障害特別調査報告から，ソフトウェア産業の成熟度は？

報告書概説のポイント

1. 事象は高橋さんからの報告と重複するため割愛
2. ソフトウェア保守の観点からポイントを絞り概説
3. システム運用面の問題は簡単な紹介程度に留める
4. 経営管理・組織管理面はできるだけ丁寧に説明
5. 現行ソフトウェアに対する扱いがポイント
6. 最後に本報告書（含むMHBKの再発防止策）に対する私見も入れる

   ⇒大島さん、弘中さんからの報告と異なる（または重複する）部分もあるが、議論活性化のため

MHBKシステム障害特別調査委員会とは？

- 2011年3月14日～24日に渡り発生した大規模な勘定系システム障害に対し，4月11日に設置され，弁護士，公認会計士，ITコンサルタントで構成された第三者委員会。
- 委員会作業補助者
  - アンダーソン・毛利・友常法律事務所
  - 有限責任監査法人トーマツ
  - アクセンチュア株式会社

委員会の目的と調査方法

調査目的

①本システム障害の原因究明

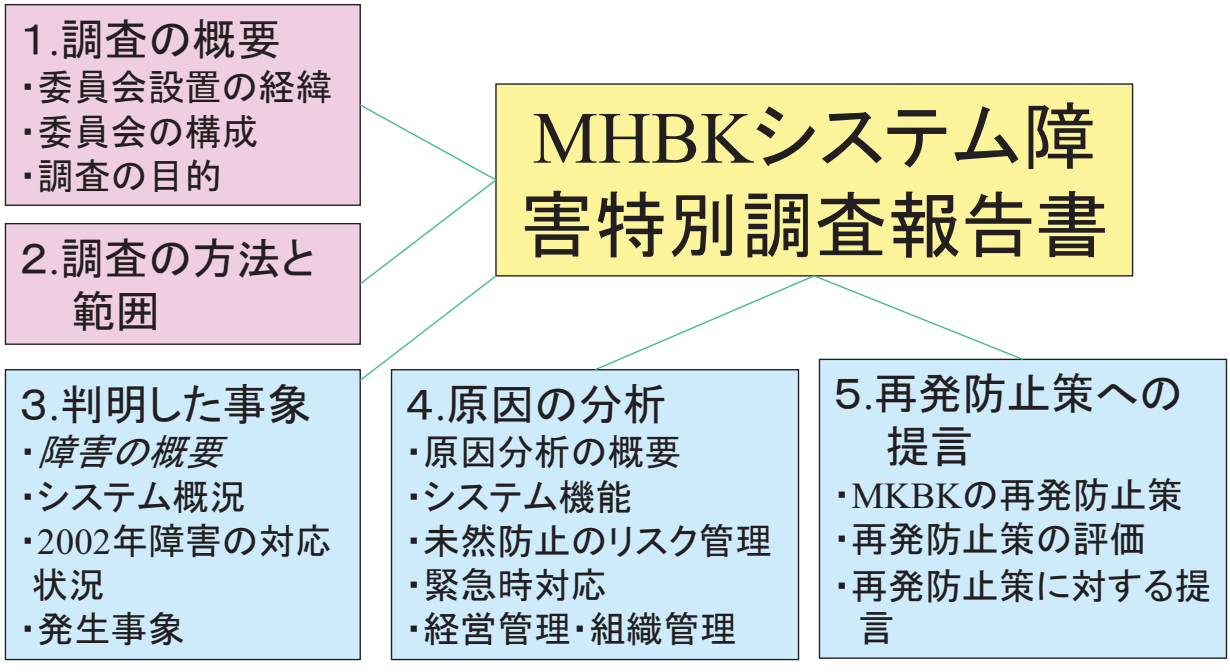
②MHBKが実施する再発防止策の妥当性の評価・提言

調査方法

①関係文書（内部資料，監査報告書，諸規程，契約書等）の検証分析

②関係部門・関係会社の担当者等ヒアリング（47回）

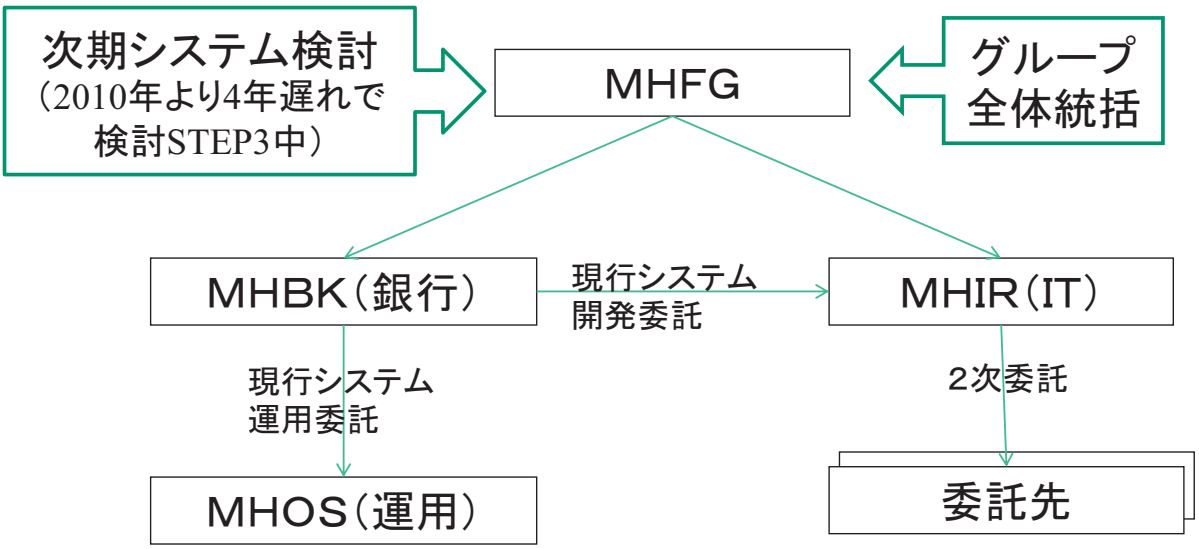
調査報告書の構成



システムの概況

1. 現行システムは旧FBのデータを旧DKBのシステムに移行（片寄せ）したもの（2004年12月完了）。
2. 本障害は主要勘定系システム（STEPS）で発生。
3. STEPSは日中オンラインと夜間バッチの同時実行を想定せず（時間を分けてスケジュール）。
4. 朝，夜間バッチが終了すると翌日の夜間バッチを作成するDJS切替を行い，日中オンライン処理開始。
5. 夜間バッチの終了が遅れると，DJS切替実行を遅らせ，日中オンラインの開始を遅らせるか（ケース1），夜間バッチを中断しDJS切替を強行し，夜間バッチの未処理を手動実行する（ケース2）しかない。

システム開発・運用の役割



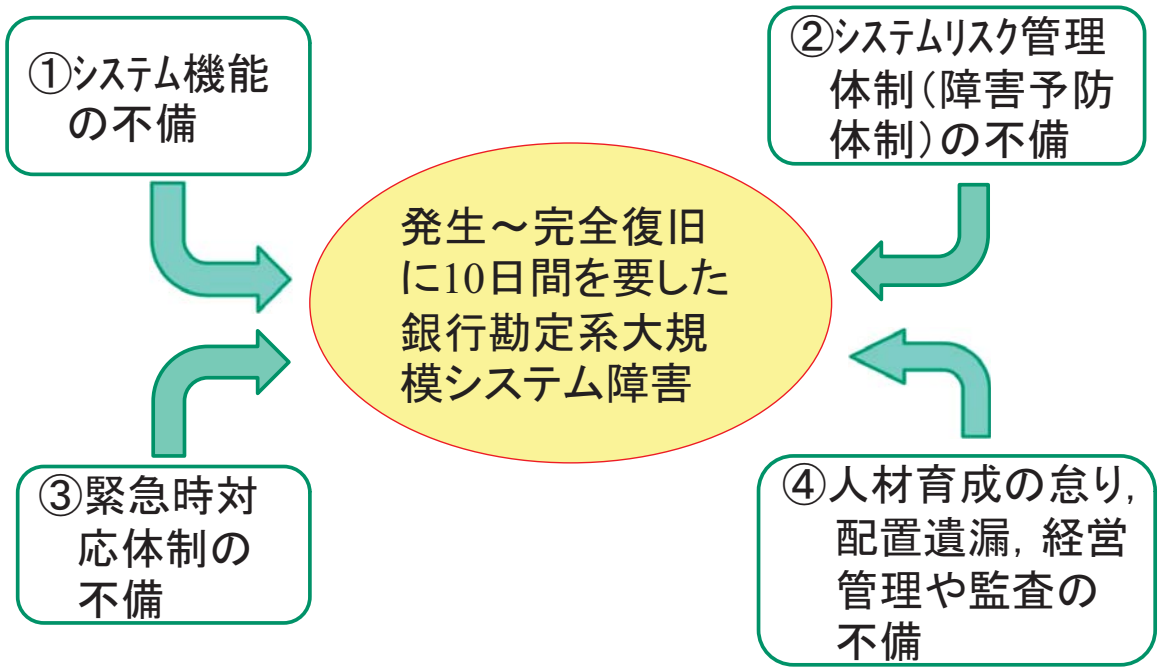
関連する主な規程

- 事業継続管理関連規程
  - システムコンティンジェンシープラン
    個別：システム復旧マニュアル，オフサイトバックアップ復旧マニュアル等
    シナリオ別：センター被災想定資料　等
  - 業務別ビジネスコンティンジェンシープラン
    想定ケース別・対応業務別規程
  - 訓練規程（各コンティンジェンシープラン別）
- その他関連規程
  システムリミット値点検規程

緊急時態勢と2002年障害対策実施状況

- 緊急時対応態勢
  MHBKの頭取，IT・システム統括役員，IT・システム部長が各役割を基に，担当者への指示，関係者への報告を行う
- 2002年4月発生システム障害の再発防止策
  - ・システム統合PJの管理体制の強化など30施策
  - ・システム監査の強化
- 対策実施状況
  - ・システム統合PJは統合が済み解散
  - ・システム監査：その後重大障害発生なく，主要勘定系の監査ランクを下げていた

本障害発生の原因



①システム機能の不備

- STEPSは1988年（凡そ4半世紀前に）稼働。
- 夜間バッチ機能のリミット値超過時対応機能の不備
  - リミット値内での繰返し処理なし
    ⇒リミット値を超過すると異常終了
  - 異常終了したとき，データの欠落が発生
- 夜間バッチ終了遅延時対応機能の不備
  - オンラインとの並行稼働不可
  - バッチ処理終了が遅延したとき，方法は次の2通りのみ
    1）バッチ終了までオンライン開始を遅らせる
    2）バッチを強制終了し，残りを手動で起動

②システムリスク管理体制の不備

- 夜間バッチのリミット値の見直しが実施されず
- 同リミット値はMHBKのシステムリスクCSA点検票に無し
- 他行や他社の類似システム障害を参考にせず
- 大量振込集中時の対応検討が不十分
- 携帯電話による振込処理導入時，高負荷テスト実施せず（理由：「システム開発」を伴わず導入できたため）
- ユーザ部門から振込処理集中時の処理に問題はないか？との確認依頼があったが，それを判断できる担当者に依頼が廻らず

③緊急時対応体制の不備

- 緊急時態勢の問題を事前にチェックする実効性あるプロセスなし
- 2002年4月のシステム障害の再発防止策不十分
- それでも同対策でシステム障害が減少し，油断！
- 最悪リスクシナリオの検討不十分
- MHBKとMHIRの連携不十分（実担当への連絡遅れ）
- IT・システム統括部へのエスカレーション遅れ
- MHBK，MHIRの統括機能不足
- 夜間バッチ突抜け時の対応パターン検討不足
- 各種手順書の実効性確認不足

④人材育成，配置遺漏，経営管理，監査不備

- 障害の影響度を分析でき，多重障害の復旧見通しを立てられる人材が不足（*火消し棟梁の不足*）
- 関係会社間で障害発生時対応訓練実施されず
- 長期安定稼働機能の可視化が衰退（*ﾌﾞﾗｯｸﾎﾞｯｸｽ化*）
- 次期ｼｽﾃﾑ検討中。現行ｼｽﾃﾑの整備に注力不足
- MHIRは再委託で空洞化。精通した技術者減少。
- 開発及び運用の自動化による実務経験不足
- 既存システムの知識・技術・ノウハウの継承不足
- 内部ｼｽﾃﾑ監査体制の形式的実施，実効性希薄
- 外部ｼｽﾃﾑ監査が実施されていない
- ﾋﾞｼﾞﾈｽ環境の変化や利用者の多様化に対応できず

MHBKの再発防止策

(1) 今回と同様のシステム障害の再発防止策
　ア「障害発生の未然防止の取組み」に関する改善・対応策
　　(ア) 大量取引が想定される口座への手続き見直し・管理・徹底
　　(イ) 預金センター集中記帳におけるリミット値見直しと大量データ監視
　　(ウ) 商品ごとのリミット値を意識した運営の確立とリスク統制
　　(エ) 預金センター集中記帳における設計・仕様の改善対応策
　　(オ) 新商品・サービス開発時のシステムリスク評価のレベルアップ
　イ「障害発生後の不適切な事後対応」に関する改善・対応策
　　(ア) センター集中記帳における時限の設定と影響等の明確化
　　(イ) 異常終了に繋がる大量データへの対応
　　(ウ) センター集中記帳が遅延した場合の対応案の整理
　　(エ) 重要な決済業務を中心として優先すべき業務の範囲・対象を選定し、各対応策、影響、制約事項、作業優先順位等を具体化、手順の策定
　　(オ) 人材強化に繋がる計画的な訓練の実施
　　(カ) 緊急時におけるノウハウ・経験を有する人材の招集体制の確立
　　(キ) 緊急時に有効に機能する態勢の見直し
　　(ク) 正確な情報に基づく適切な営業店指示及びHP等への啓示
　　(ケ) 苦情等の把握・分析や対応・改善策の検討
　　(コ) システム障害による実費・損害賠償等費用負担における公平な対応
(2) システムリスク管理態勢に対する改善策
　ア システムリスクCSAのレベルアップ
(3) 事業継続管理態勢に対する改善策
　ア 緊急事態発生時の対応態勢の改善
　　(ア) 緊急時行内体制の見直し
　　(イ) 緊急事態発生直後の情報連絡・共有フローの見直し
　　(ウ) 役職員を対象とした緊急時対応をテーマとする研修の実施
　　(エ) システム障害を想定した全行訓練を通じた実効性の検証
　イ システムコンティンジェンシープランの改善
　　(ア) システムコンティンジェンシープランの記載内容明確化
　　(イ) システムコンティンジェンシープランの実効性向上に向けた疑似体験訓練の実施
　ウ ビジネスコンティンジェンシープランの改善
　　(ア) ビジネスコンティンジェンシープランの再点検及び見直し
　　(イ) ビジネスコンティンジェンシープラン発動時の留意事項の徹底
　　(ウ) 必要なビジネスコンティンジェンシープラン対応についての営業店周知と訓練の実施

MHBK再発防止策の調査委員会評価

1. 再発防止策の評価
   ⇒未然防止管理態勢は評価の実効性を高めるための工夫，新サービス導入時のリスク分析（システム開発を伴わない場合）
2. 早期復旧に向けた管理態勢⇒評価できる。
3. 経営管理及び組織管理
   ⇒人事施策は評価できる。
   ⇒内部監査は監査手順の見直し要
4. 再発防止策に対する提言
   ⇒システム機能は今回以外に同様の問題点がないか確認し，コンティンジェンシープランのリスクシナリオに盛り込め
   ⇒未然防止の管理態勢はCSAのチェック項目のレベルアップと精度向上（他部門のクロスチェック）
   ⇒早期復旧の管理態勢は総合的な指揮命令系統確立，システムコンティンジェンシープランの定期点検，有識者活用

MHBK再発防止策の調査委員会評価

4. 再発防止策に対する提言（続き）
   ⇒経営管理及び組織管理
      →短期的：訓練の繰り返し
         長期的：MHBKとMHIRの人材交流の活性化，計画的な人材育成
   ⇒監査は監査リストの見直し，グループ監査体制見直し，外部監査活用等
5. 将来提言
   ⇒再発防止策の継続した実施
      ｼｽﾃﾑ統合の早期実現

ソフトウェア保守から見た本報告書の評価

- 報告書にソフト保守が触れられていない理由は？
- 開発を伴わないときの対応整備は検討されているが，小改造（修正）の対応手順が本当に今のままで良いか検証無し。
- 「早急にMHBKとMHCBとのシステム統合」をという提言は疑問。本障害は既存システム対応の問題。システム統合が再発防止に大きく貢献する保証無し。
- システム統合が主要対策になると，その間現行システムの環境変化への対応が後手になる可能性あり。
- 現行システムを使い続ける期間を明確に提言すべき。
- その間，優秀な保守技術者を投入し，現行システムを使い続ける場合のリスクとメリットを明確にすべき。

参考（ソフト保守の考え方。各社Webページより）

MHIR：

ISO9001:2008 品質マネジメントシステム（JQA-QM4427）

銀行システムグループが受託するソフトウェアの設計・開発

銀行システムグループ、人事部、調達管理部

MRIDCS:

CMMI レベル4

ソフトウェア開発プロセスの能力成熟度を評価する公式アプレイザルにおいて成熟度レベル4を達成

達成日 2011年9月22日 公開終了日 2014年9月22日

適用モデル CMMI-DEV v1.2A 適用評定手法 SEI SCAMPI v1.2A

適用範囲 全システム開発部門（含む：東北DCS）

主な業務内容 アプリケーション開発・保守業務全般

みずほ銀行　システム障害レポート　一覧

2012年4月10日
「仕事品質」改善教室　大島道夫

(1)システム機能上の不備

【環境の変化に対して「要件」の見直しをしていなかった】

MHBKの勘定系システムは1988年に稼動を開始
↓
ATMは、当時においては稼動時間が限定されていた
↓
現在では24時間の利用が可能となり、インターネット・モバイルを始めとした取引チャネルが多様化し、情報量が増大したばかりか短期集中的な処理を求められるようになった。
↓
システム復旧に充てられる時間的な余裕は減少しつつある。したがって、このような状況の変化に応じシステムにおいても柔軟に対応すべきであった

(1)①システム処理単位の不備
取扱うデータ量が著しく増加した現時点では、大量取引の集中に対して柔軟に対応できるように、システム機能においてリミット値を踏まえた処理の分割を図るべきところ、あらかじめそのような措置を講じなかった。

(1)②システム運用設計の不備
しかし、DJS切替を行うと、残りの夜間バッチが手動となり、その処理に膨大な手数を要することとなるばかりか、為替データの作成・送信が夜間バッチの後に一括して行われることにより為替送信も遅延する仕組みとなっていた。
したがって、担当者がこのようなSTEPS、TARGETの基本的な仕組みを理解し、夜間バッチ突き抜けの場合のリスクが大きいことを認識すべきだった。

(2)システムリスク管理態勢上の不備

【環境の変化に対して「リスク評価」の見直しをしていなかった】

当該リミット値はシステム稼動時から設定の見直しはなされておらず、定期的な点検項目にも入っていなかったため、担当者において、夜間バッチにおける当該リミット値が存在することの認識すら不十分であった。

(2)①定例リスク評価の不備
今回の障害と類似ケースである大量振込取引に関する障害事例、夜間バッチ遅延に関する障害事例が紹介されていたが、十分に活用されることなくシステムリスク点検項目の見直しがなされていなかった。

(2)②初期のリスク評価の不備
・大量データのリスク評価漏れ
　ルールとなっているチェックリストの未使用→テスト漏れに繋がった。
・問い合わせ窓口が曖昧
・問い合わせ内容の重要性/緊急性の誤認識

(3)復旧対応における緊急時態勢の不備

【事前に準備した規程やプロセスが機能していない】

事業継続管理関連規程として事前に準備されていたにもかかわらず、
・緊急時における態勢が実効性を伴っていなかったこと。
・システムコンティンジェンシープランとして想定すべき事象が不足していたこと。
・復旧対応の手順書が実効性を伴っていないこと
・チェックプロセス及び訓練が、実効性を検証する役割を果たせなかったこと。

(3)①緊急時体制/指揮命令系統の不備
緊急時における、MHBK、MHIRの経営陣間の指揮命令系統が不明確であったこと
・緊急時態勢に適切な判断がされなかった。
　・リスクシナリオの検討が不十分
　・適切な情報連携がなされなかった
　・統括機能が不足していた
・障害対応時の情報連携無し、行動遅延
　・大量データ発生情報が、ユーザー部門からIT部門に引継ぎされていない。
　・要員投入が遅れた。

(3)②想定すべ事象の不足
決済業務を担う最重要システムであるにもかかわらず、障害についての事象の想定が不足していた。またシステムコンティンジェンシープランが整備されていない。

(3)③手順書の実効性不足
二つの手順書が時間的考慮を踏まえた手順書になっていなかった。作業内容に関する不正確な見積もりに基づき作業の実施を判断することとなった。
「CMFセンター記帳ABEND時対応について」
「STEPS夜間バッチ突き抜け時のオンライン対応」

(4)経営管理及び監査の不備

【経営陣→最重要システムのリスク認識の甘さ】
【監査→監査の形骸化、経営者に提言する弱さ】

(4)①人材計画/適所配置の不備
・勘定系システム全体への影響を分析する能力を有した実務人材が不足していた。
・多重障害の復旧見通しが立てられる実務人材が不足していた。

<情報不足での判断ミス>
結果として、発生した事象に関する断片的な情報にとどまり適切な障害対応の判断には不足していたにもかかわらず、そのまま受け入れる判断しかできなかった。

<マネジメント人材不足>
一連の障害を通じて、システム全体を俯瞰でき、かつ、多重障害の陣頭指揮を執り得るマネジメントの人材も不足していた。

<訓練未実施/訓練への取り組み姿勢は?>
システム障害を想定した実地訓練がこれまで実施されていなかった

<若手への要件、仕様スキル伝承の不備>
「ノウハウや仕様の可視化」への対応の遅れ

<障害対応経験不足>
品質向上への取組みの結果、勘定系システムにおいて障害件数が減少し、現実の障害に対応する経験を積む機会が減少した。

<対応策の先送り>
既存の勘定系システムの設計全体を見直す取組みを次期システム構築時に実施する方針であること

<再委託によるリスク認識の低減>
システム開発がグループ外部へ再委託されること

<自動化による人の対応力脆弱>
更に、開発及び運用の自動化の推進が進められたために、システム部門の人材が自らシステム開発やシステム運用の実務を学習する機会を減少させたこと

<人材の流失/退職>
加えて、経験豊富な職員の退職と銀行業務の実務経験を有する人材の割合が急速に減少

(4)②監査の実効性の不備

・システム監査が不十分
・グループとしての監査体制の不備
・外部監査の活用の遺漏

いずれの問題も、内部監査部門による活動が十分に行われていない。

<調査資料と記述について>
みずほ銀行のサイトに掲載されているシステム障害特別調査委員会の調査報告書(2011年5月20日付け)
(1)～(4)の見出しおよび記述は、同調査報告書より転記
(4)の**<青文字>**部分は、大島加筆

<コメント>

(1)トラブルの影響度とこれにかけるコストバランス
　市場への影響が大きい保守のトラブルの予防は、経営判断によるところが大きい。(経営責任)
　①安心安全のシステム開発と運用の必要性はわかるが何処までもコストをかけられない。ここの経営判断が肝である。
　②大きな組織になると緊急時の対応も大きな仕組みとなり、効果が無い手順となり、形骸化する。システムは、大規模になっても、小回りのきく緊急手順にすることが肝要である。大規模悪にも繋がる。
　③有能な人材を継続的に配置する必要性はわかるが労務費の負担とモチベーション維持が課題となる。
(2)監査
　①トップへの提言は、第三者の立場でも、かなり勇気が必要である。
　②問題を直言できない曖昧な監査報告書では、意味が無い。しかし、直言した報告書でも経営者の認識が甘ければ、これも意味が無い。
　　該当するシステムには○%の維持コストをかけるという法規制が必要か?
　　受ける方に無理のない保守契約をきちんと結んで成果が出せるようにすべき！
(3)システム/ドキュメント関係
　①システム設計考慮不足のミスより「見直していない」ミスの方が影響が大きい。実現的な対策は、これか?
　②許容範囲やリスクを想定して設計してもすぐに陳腐化する。それよりも見直しが大切、またコレしか対策が無いか?
　③起きてもいない緊急手順書は書けない。想定した範囲は、いつも想定外となる。
(4)トラブル報告書
　・第三者が形式的に作成した報告書では、現場の改善に役立つ本当のことが書かれていない。
　・今回のシステム障害レポートも言葉が整理され、まとめられているが故に「現場の声」が消えてしまっている。
　・米国の司法取引制度等を採用する。(本当のことを言えば、責を軽くするまたは問わない)
　　形式的な報告書より、次のミス防止に役立つ報告書が必要。
(5)訓練への取り組み姿勢
　①訓練が必要だとしているが、本当に本番で出来るのか→影響を小さく出来るが、ゼロにすることは出来ない。
　②障害が削減するとリスクへの認識も甘くなるのが人情→訓練でどこまでカバーできるのか?我々も火災訓練での行動を顧みれば一目瞭然である。
　　トラブルを繰り返さないと本当にわからないのが人間か?　「あの時のトラブル」を継承することはできない。
　　いつか忘れる。

<結果>

(1)このシステム障害事例が糧となるように、我々も勇気をもって職場点検をしよう！
　・ソースばかりに着目せず、視野を広くして問題認識を持とう！　業務視点/お客様視点
　・そして、改善点やリスクを勇気を持ってエスカレーションしよう！　あきらめずに！
　・お客様/社内で行う訓練には、本気で取り組もう！
(2)我々現場レベルで本気で内部監査を実施しよう！
　・また内部監査の工数は、お客様にも社内でも認めてもらえるように提案しよう！
(3)次のことに本気でコスト投資してもらえるように提案しよう！
　①まず保守要員の価値を再認識しよう！
　　給与アップとポスト提供、そして休暇取得実現
　　→すれば情報も提供するし、要員自ら動く！
　②要件、機能等の見直し活動の実施
　③リスク評価の仕組み強化、評価者の育成
　④緊急時の手順書見直し活動
　⑤正直な報告書がかける監査員育成
　※②から⑤は、利益に直結しないので法規制も必要か?
(4)小さなトラブル報告書も担当者の気持ち、どう判断したのか、等の本音の報告書を書こう！　そしてその改善策迄、文章化して残そう！

SERCフォーラム

大規模障害に学ぶソフトウェア保守
～ソフトウェア保守者の見た　みずほ銀行オンライン障害～

SERCの見解１
組織論、マネジメントの観点から

2012年5月18日

SERC研究員　大島　道夫
SERC研究員　弘中　茂樹

目次

1. はじめに・・・
2. 考えるよりどころ・・・
3. 障害レポート一覧の解説
4. 一言で言うと・・・
5. 私たちでやること
6. Q&A

<ご参考資料>

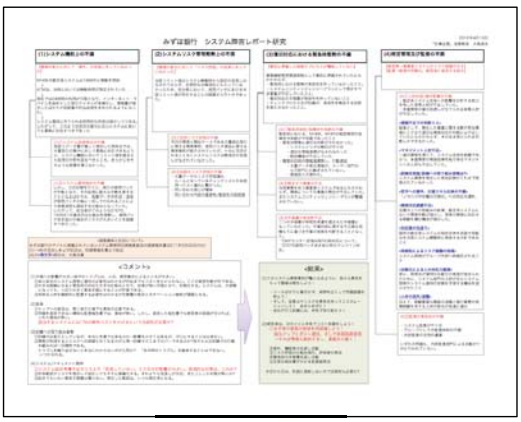
1.はじめに・・・

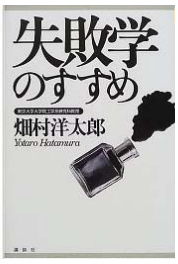
- 私たちは、みずほ銀行オンライン障害から何を学ぼうとしているのか?

- ✓ 誰が悪いのか?
- ✓ 何が悪かったのか?
- ✓ ここがまずかった!
- ✓ こうすべきだった!
- ✓ こうなるとまずくなるという学び?
- ✓ 私たちの職場の改善につながる「気づき」の学び?

2.考えるよりどころ・・・

みずほ銀行オンライン障害
レポートを整理してみた。





工学院大学教授
<畑村 洋太郎:著>

保守業務改革のために
学ぶべきものは、何か?

3.障害レポート一覧の解説

- お手元の「みずほ銀行システム障害レポート一覧」の内容を解説いたします。

4.一言でいうと・・・

- 小さな失敗(ミス)を避けることをしてきたため、将来起こりうる大きな失敗の予防が出来ず、また失敗の対策も後手となった。

人とコストを軽視/立場重視した対策は、
いざという時に役に立たない！

5.私たちでやること

職場点検

(1)このシステム障害事例が糧となるように、我々も勇気をもって職場点検をしよう！

①ソースばかりに着目せず、視野を広くして問題認識を持とう！　業務視点/お客様視点
②そして、改善点やリスクを勇気を持ってエスカレーションしよう！　あきらめずに！
③お客様/社内で行う訓練には、本気で取り組もう！

上位へ提案

(3)次のことに本気でコスト投資してもらえるように勇気を持って提案しよう！

①まず保守要員の価値を再認識しよう！
給与アップとポスト提供、そして休暇取得実現
→すれば情報も提供するし、要員自ら動く！
②要件、機能等の見直し活動の実施
③リスク評価の仕組み強化、評価者の育成
④緊急時の手順書見直し活動
⑤正直な報告書がかける監査員育成

本気の内部監査

(2)我々現場レベルで本気で内部監査を実施しよう！

①また内部監査の工数は、お客様にも社内でも認めてもらえるように提案しよう！

使える報告書

(4)小さなトラブル報告書も担当者の気持ち、どう判断したのか、等の本音の報告書を書こう！　そしてその改善策迄、文章化して残そう！

6.Q&A・・・

End of Presentation

<ご参考> 10年後に必要なエンジニアは?

「こうすれば、うまくいく」

「どこかで見聞きした方法である」

「知識」

「正しいやり方」

「標準プロセス」

「新規開発」

「大規模対応可」

「Vモデル」

パターン化された既成/新規の
対応には強いが、「自分たちで
考えることや創意工夫」は苦手。

ミスは、命取り！
できるだけミスを避ける行動をとる。

「こうなると、まずくなる」

「失敗から学んだ方法である」

「智恵」

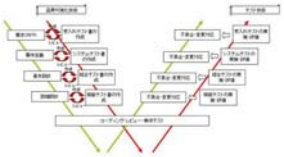
「妥当なやり方」

「改善プロセス」

「保守」

「小規模対応」

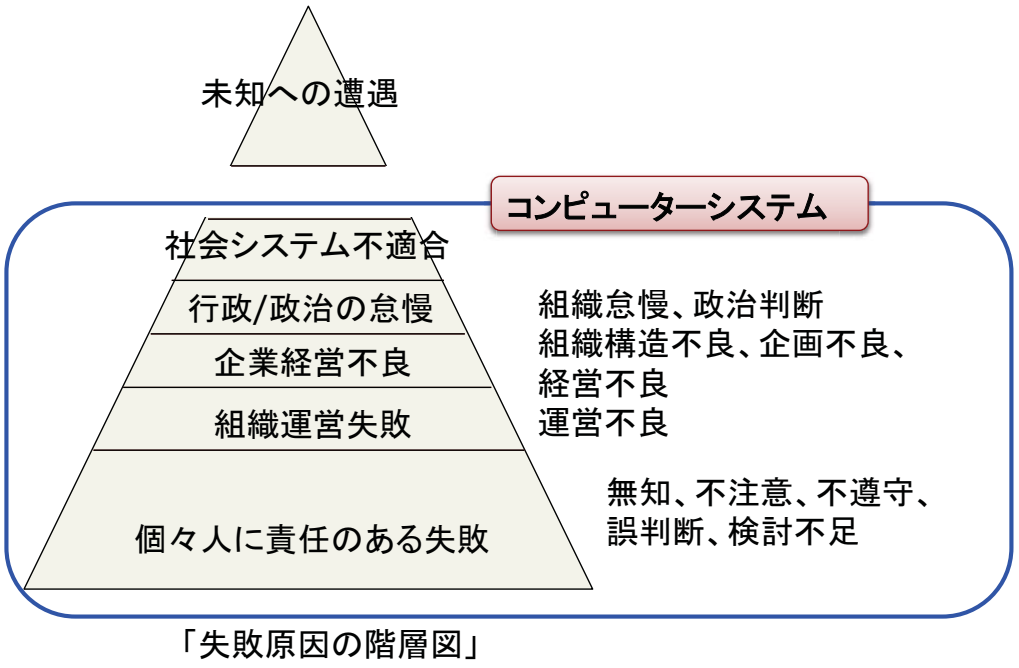
「Wモデル」



テストのバグ分析から得られた
ノウハウを前工程にフィード
バックすることで「自分で考え、
創意工夫する力」が身につく。

失敗(バグ)を成長の糧に！

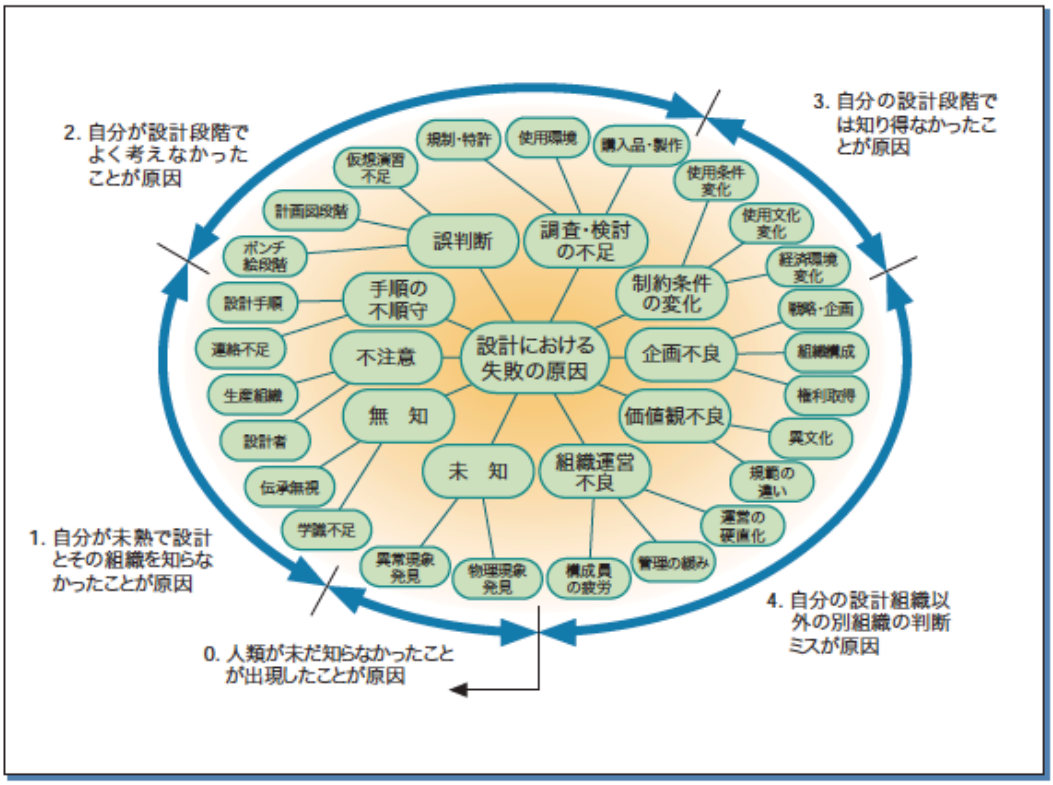
<ご参考> 「失敗原因の階層図」



「失敗原因の階層図」

(出典:失敗学のすすめ　畑村洋太郎:著)

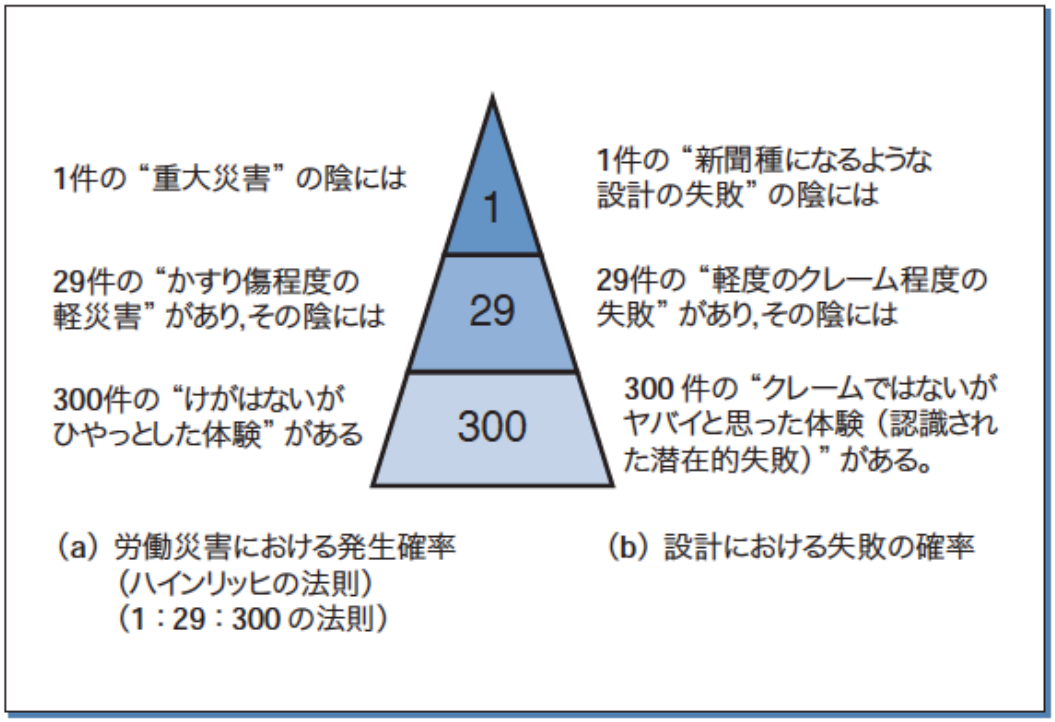
<ご参考> 設計における失敗原因の分類



図−3　設計における失敗原因の分類

(出典:失敗学のすすめ　畑村洋太郎:著)

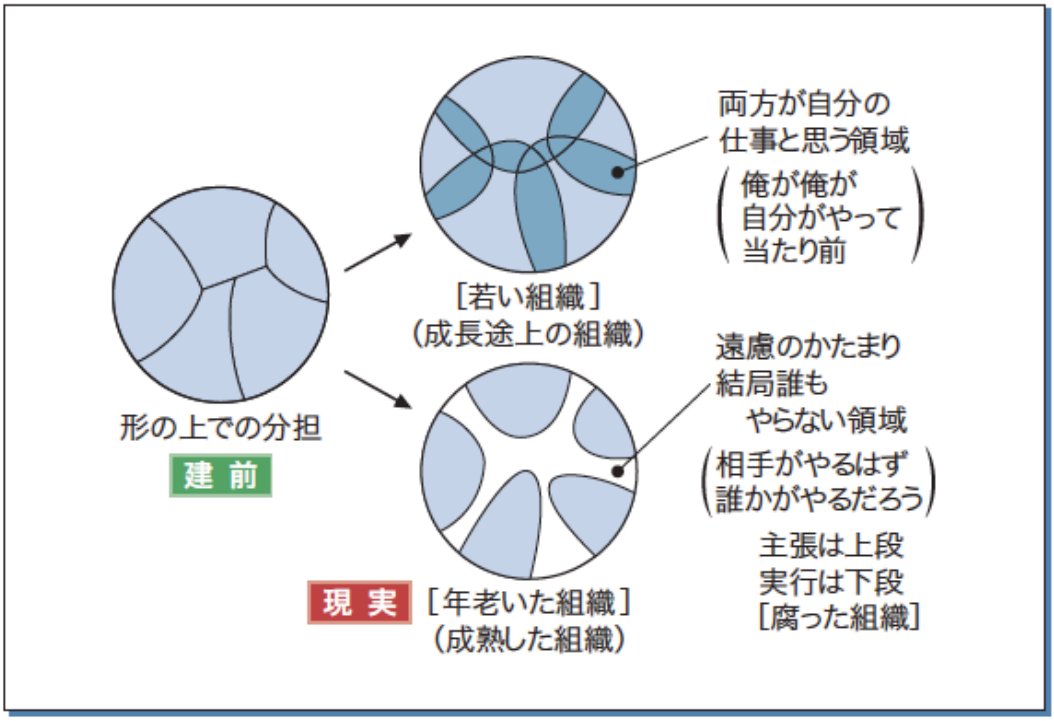
<ご参考> 設計における失敗の顕在化の確立



図−2　設計における失敗の顕在化の確率
(労働災害の発生確率から類推)

(出典:失敗学のすすめ　畑村洋太郎:著)

<ご参考> 組織の中での役割分担と実際



図−1　組織の中での役割分担と実際

(出典:失敗学のすすめ　畑村洋太郎:著)

大規模障害に学ぶソフトウェア保守

～ソフトウェア保守技術の観点より～

2012年5月18日
SERC　Dグループ　高橋　芳広
E-mail：yoshi-tk@acm.org

この発表の目的

■ 自己紹介も兼ねて

約30年のソフトウェア開発・保守の現場の代表

●企業内での障害の反省
　技術的な視点が主→経営的、管理的な面の反省は少ない
●社会的な障害の反省
　経営的、管理的な反省が主→技術的な情報の共有は少ない

⇒技術的な面での情報共有が必要
　※同じ失敗も多い（例　携帯メールの漏洩）
　　→　ソフトウェア版失敗百選

二つの視点

今回にも共通する

「ソフトウェア保守をやっているとよく耳にする話題」

■ 保守の範囲とは？

「（前略）当該サービスにおいては**既存システムで対応可能で有り、システム開発を行う必要がなかったため、**両部門協議において、B社のシステムとMHBKシステムの接続部分のテストのみ実施することになった。（以下略）」

~「調査報告書」システム障害特別調査委員会　P23　（2）新商品導入時のリスク評価より抜粋

ソフトウェアの修正が発生しなければ、保守ではないのか？

■ 「上限値」を考える

上限値はシステムを保護するためや、必要とするリソースを見積もるために必要

→だが、ひとたび上限値超過が発生するとよりややっこしい事態が

※エラー処理は正常処理に比べて、より時間がかかり、運用も複雑

文献による保守の定義

■ JIS X0161:2008「ソフトウェアライフサイクルプロセス－保守」

●ソフトウェア保守（software maintenance）

ソフトウェアシステムへの費用対効果の高い効率的な支援の提供を要求するあらゆる活動。活動は，引渡し後と同様に引渡し前にも実施される

●ソフトウェア保守プロセス主要アクティビティ

- プロセス実装
- 問題分析及び修正の分析
- 修正の実施
- 保守レビュー及び受入れ
- 移行
- 廃棄

※ソフトウェアの修正がない場合を扱うかどうかは、明示されていない

ソフトウェア保守のタイプ（JIS　X0161:2008より）

- **適応保守（adaptive maintenance）**

  引渡し後，**変化した又は変化している環境において，ソフトウェア製品を使用できるように保ち続けるため**に実施するソフトウェア製品の修正

- **是正保守（corrective maintenance）**

  ソフトウェア製品の引渡し後に発見された問題を訂正するために行う受身の修正（reactive modification）

- **完全化保守（perfective maintenance）**

  引渡し後のソフトウェア製品の潜在的な障害が，故障として現れる前に，検出し訂正するための修正

- **予防保守（preventive maintenance）**

  引渡し後のソフトウェア製品の潜在的な障害が運用障害になる前に発見し，是正を行うための修正

ソフトウェア標準には、**ソフトウェアの修正がない場合が含まれてはいない**
⇒提案してはどうだろうか！

上限値の起源

- **有限な資源による**
  - ・メモリ、ディスクなど利用できるハード資源が有限であるため制限
  - ・データやサービスの整合性を保つために、資源を確保の処理でエラーになる前に制限をかける

- **処理時間の制限**
  - ・サービス時間の制限（決められた時刻までに処理を終わらせる）

- **アルゴリズムの制限**
  - ・採用したアルゴリズムに起因する
    例えば、処理するデータの量と処理時間が非線形の場合、
    あるデータ量を超えると急激に性能が劣化する。
    このポイントを上限値にする

上限値にまわる障害の過去の事例（その1）

■ 現象

ネットワーク管理ソフトで、**新規ルータ追加時にファイルの作成失敗で異常終了した。管理対象のリソースの数は、上限値に遙かに少なく、**ディスクの容量も十分余裕があった。

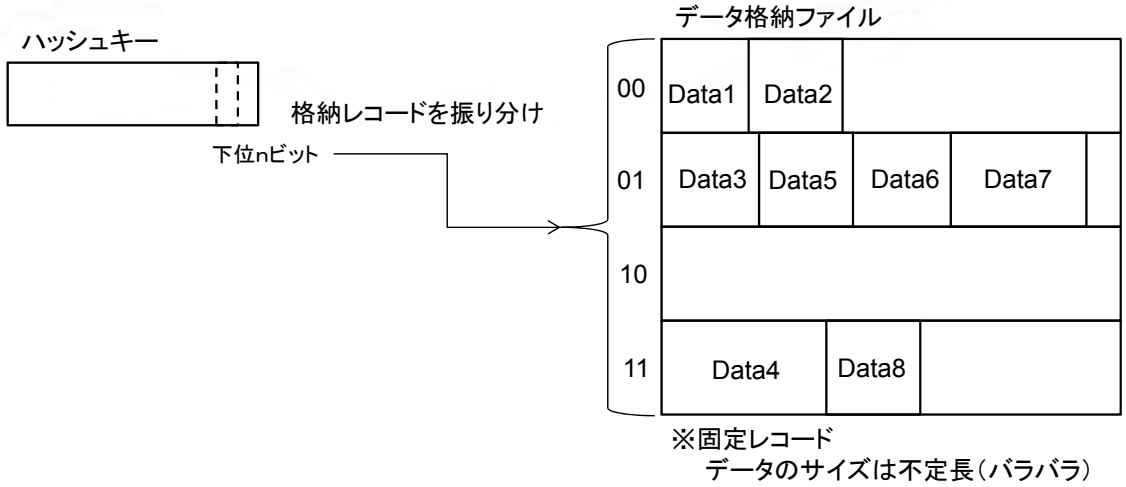
■ 原因

ネットワークリソースを管理するアルゴリズムに不備があり、特別な条件の場合管理対象の数に関わらずファイルサイズが増大した。

上限値にまわる障害の過去の事例（その2）

■ 問題のある管理アルゴリズム（1）

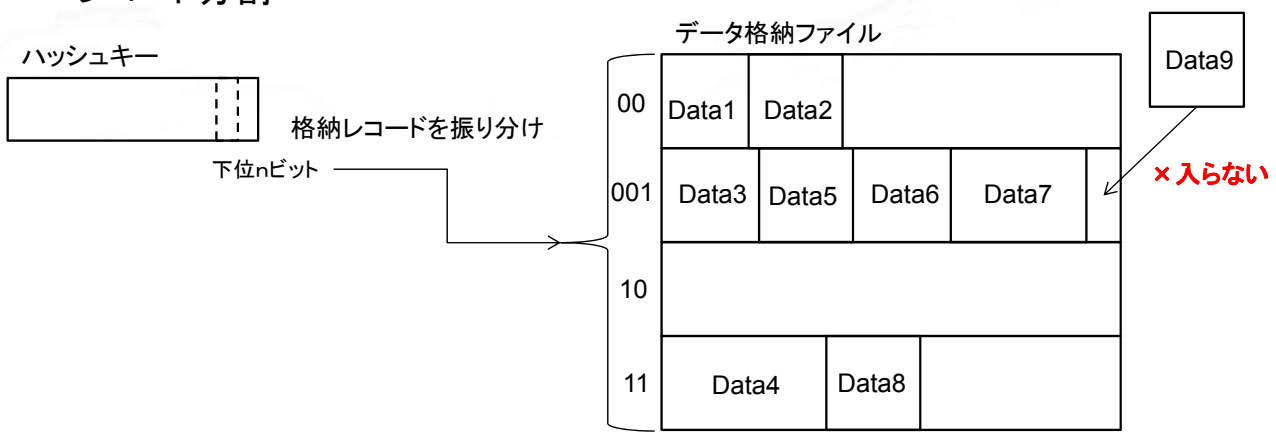
ハッシュアルゴリズム



上限値にまわる障害の過去の事例（その3）

■ 問題のある管理アルゴリズム（2）

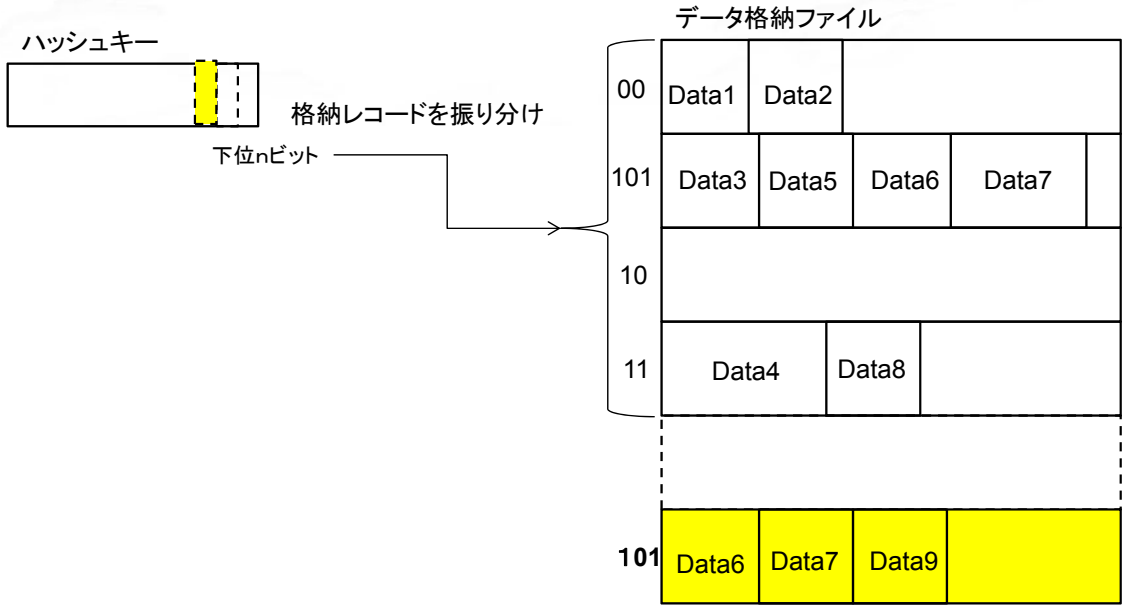
レコード分割



上限値にまわる障害の過去の事例（その3）

■ 問題のある管理アルゴリズム（2）

レコード分割

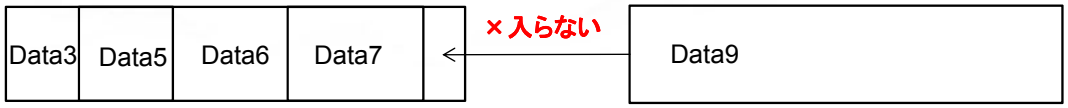


上限値にまわる障害の過去の事例（その4）

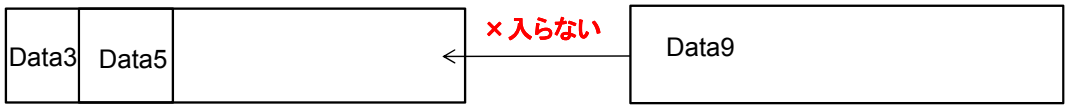
■ 問題のある管理アルゴリズム（3）

レコード長に近いデータ格納時の分割

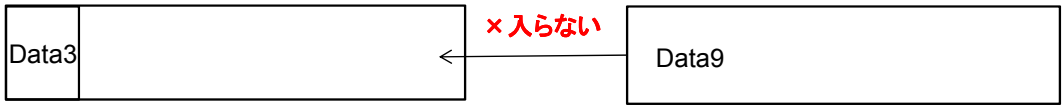
（1）分割前



（2）1回目の分割後



（3）2回目の分割後



※レコード分割をする度にファイルサイズが倍になる
（格納データは一切増えない）

上限値にまわる障害の過去の事例（その5）

■ 対処

ルータの回線の付属する情報（文字列）の上限値がレコードサイズになっていた。上限値を超えたものは破棄する（あまり重要出ない情報）。**この上限値をレコードサイズの1／2に変更（暫定対処）**

■ 教訓

・システム仕様の上限 ≠ リソースの上限
→アルゴリズムに依存することを忘れない

・リソースの容量不足による障害の原因は必ずしもハード容量不足でない
→プログラムバグで発生することもある（ことの方が多い）
例）メモリ不足が発生した場合、
扱うインスタンスの量が超えたことだけでなく、
メモリの解放漏れやメモリ確保のアルゴリズムを疑ったほうが
解決は早い

上限値超過の発生要因

■ 異常ケース

ハードウエアやソフトウェアの誤動作により発生したデータにより上限値を超える場合（想定外か、異常として想定）

・様々なアルゴリズム実装上のプログラム不良

・他システムとのインタフェース不良

→システムを保護するために、上限値を超えたデータは破棄

■ 正常ケース

サービスの集中など発生する確率は極めて低いが、システムに異常が発生しなくても上限値を超過する場合（元々起こる可能性はあるもの）

・ハプニングや災害など、特殊な事情の発生によるサービス要求の集中

・サービス要求が確率分布に従うイベントの希なケース

→上限値超えが発生してもデータは捨てられない

→上限値を超える前に処理の受付を制限する方式が望ましい

上限値を超えたときの処理1（受付抑止）

■ 受付抑止

＜概要＞

サービスの上限値を超過する場合、新しいサービスの受付を抑止

＜主な採用ケース＞

・正常ケース

＜利点＞

・データの正常性を保つのが容易

・運用の負荷が少ない

＜欠点＞

・システム全体を通した制限値の整合性を確保する設計が要求される

・サービス要求の集中ポイントをシステムの外にだすことになるので、例えば、利用者の不満になる

上限値を超えたときの処理（異常終了）

＜概要＞

上限値を超えた時点で異常終了し、 システムの資源を見積もった上で上限値を増加させ再起動する。

＜採用ケース＞

・異常ケース

＜利点＞

・システム内の複数の制限値間の整合性に対しては比較的ゆるやか

・同じデータを利用しての0からの再処理なので、設計は比較的容易

＜欠点＞

・処理時間は短縮されない（異常処理の場合増加する）ため、新たにサービス要求を受け付けると、未処理の要求が増大する

・途中で異常終了した場合、リカーバリーのための運用が必要となり、運用の負担が大きい

上限値を超えたときの処理（異常終了・半正常）

＜概要＞

上限値超えで処理を中断する場合、正常に処理が済んだ要求と、未だ処理されていないもとを分ける。再処理を行う場合は、処理が済んでいないものを対象とする

例）上限値が100で、200の要求があった場合、100個目で処理を打ち切る。再処理時には101個目から始める。

＜主な採用ケース＞

・正常ケース、異常ケース

＜利点＞

・異常終了の処理よりも再処理の時間が短縮される

＜欠点＞

・処理が済んだものと、済んでないデータの整合性を確保するために

・時間に厳密なサービスの場合、不公平性が生まれる

まとめ

■ ソフトウェアの修正がない場合のテスト漏れ

システムにサービスを追加する場合でもソフトウェアの修正がない時のプロセス標準が無く。問題が発生することがある

⇒標準への提案が必要

※近年、非機能要件が取り上げられているようになった例あり

■「上限値」

システムの整合性を保つための「上限値」であるが、その設計によっては上限値超過時の事態をかえって複雑にしてしまうことがある。

・異常終了させると

→再処理のための処理時間は増大する

→リカバリのための運用は複雑になる

⇒上限値超過の発生のケース、サービス要求の継続要否を考慮したアーキテクチャ設計が必要

資料１－３

10年後の保守を考える

2012年 7 月 25 日(水)

■プログラム

13:00 - 13:20　受付
13:20 - 14:00　ウラから観た、ソフトウェア保守の過去・未来・現在
　　田中　一夫（ソフトウェア・ケンテナンス研究会　事務局）
14:00 - 14:15　質疑応答
14:15 - 15:15　Ｃグループからの提言
15:15 - 15:30　休憩
15:30 - 16:50　討議
16:50 - 17:00　クロージング

■会場：芝浦港南区民センター
　　港区芝浦 4-13-1
　　UR 都市機構トリニティ芝浦 1･2 階
　　http://www.kissport.or.jp/sisetu/sibaura/index.html

ソフトウエアメインテナンス研究会

Ｃグループ

「１０年後の保守を考える」

第２回フォーラム

２０１２年７月２５日　１３：１５～１６：３０

東京都港区芝浦港南区民センター

アジェンダ

＜フォーラム＞

13:00～	受付
13:15～13:20	オープニング
13:20～14:00	ゲストスピーチ 「ウラから観た、ソフトウェア保守の過去・未来・現在」 田中　一夫氏（ソフトメンテナンス研究会）
14:00～14:15	質疑応答
14:15～15:15	Cグループからの提言
15:15～15:30	休憩
15:30～16:30	ディスカッション
16:30～16:35	クロージング

＜フォーラム終了後＞

17:00～	情報交換会

Cグループからの提言

１．「ソフトウエア保守の自動化を目指して」

㈱ｱｲ･ﾃｨ･ﾌﾛﾝﾃｨｱ　玄間　稔

２．「１０年後の保守作業環境について」

㈱ｱｲ･ﾃｨ･ﾌﾛﾝﾃｨｱ　丸山　陽一

３．「私が考える１０年後の保守」

中央ｺﾝﾋﾟｭｰﾀｰ㈱　伊藤　順一

４．「10年後の保守担当に求められるスキル」

東芝ｿﾘｭｰｼｮﾝ㈱　佐井　由美子

５．「保守開発のグローバル化」

ｱｲｴｯｸｽ・ﾅﾚｯｼﾞ㈱　岡田　浩

６．「１０年後を見据えた人材育成」

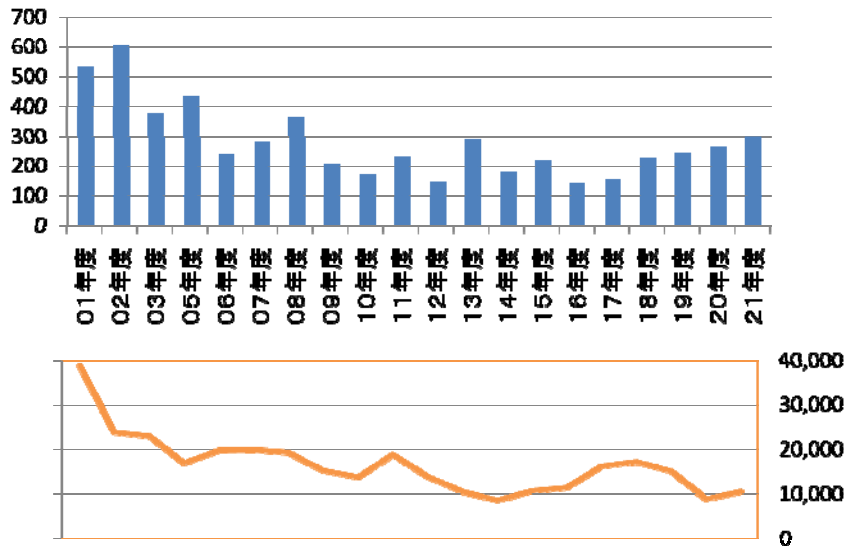
㈱ｱｲ･ﾃｨ･ﾌﾛﾝﾃｨｱ　田中　創

ウラから観た、ソフトウェア保守の過去・現在・未来

2012年7月25日

ソフトウェア・メインテナンス研究会事務局
田中　一夫

メンテナンス研究会のトレンド



研究会のテーマ（1）

01年度（1990年度）
- 要員に関する問題　ドキュメン関する問題
- 保守プロセスモデル　保守工程における保守技
- 保守性を考慮した開発のあり

02年度（1991年度）

要員問題	ドキュメント問題
作業管理問題	保守技術問題
開発方法	リポジトリ情報

03年度（1992年度）

保守要員管理に関する問題	統合化CASEツールの調査
リバースツールの調査	ダウンサイジングの保守
保守における品質管理	保守情報のリポジトリ化

研究会のテーマ（2）

03－04年度（1993-4年度）

保守情報のリポジトリー化	保守における品質管理
保守の料金体系とシステムの原価	
パッケージの有効利用と保守ツール	
保守技術者と高度情報化人材育成カリキュラム	

05年度（1995年度）

ソフトウェア保守技術者育成テキスト	
メタ情報リポジトリと支援環境	保守の生産管理開発方法

06年度（1996年度）・・・合宿地：釧路、YIS

2000年問題対応ガイド	オープンシステムの保守
ソフトウェア保守のロジスティクス	保守ツールの評価

研究会のテーマ（3）

07年度（1997年度）
- オープンシステムの保守
- SMSGの考える保守とは
- 2000年対応におけるテストの効率
- 利用者支援の改善
- 保守ツールの評価

08年度(1998年度）・・・合宿地：酒田、JFITS
- オープンシステムの保守(障害個所の特定方法)
- 保守のペーパーレス環境
- 保守作業改善の基盤技術調査(情報共有における保守作業の改善)
- SMSGの考える保守とは

09年度（1999年度）・・・合宿地：小浜
- オープンシステムの保守～顧客満足度向上策～
- 保守作業改善の基盤技術調査ー保守のプロセス改善ー
- 保守のペーパーレス環境-テストドキュメントの電子化-
- SMSGの考える保守とは

研究会のテーマ（4）

10年度（2000年度）・・・合宿地：鹿児島
- オープンシステムの保守～ビジネスアプリケーションをオープンソースする事で保守が楽になるか～
- 保守作業改善の基盤技術調査ー保守作業改善の為の技術動向、導入のポイント
- ASPの保守を考える-W ebシステム保守の考察と保守担当者満足度向上策-
- SMSGの考える保守とは

11年度(2001年度）・・・合宿地：青森
- 保守作業改善の基盤技術調査～保守の見積りと教育の現状～
- 保守のPMとKMを考える～保守ナレッジ共有化のためのeメール再利用について～
- SMSGの考える保守とは

研究会のテーマ（5）

12年度（2002年度）・・・合宿地：下関
- オープンシステムの保守～XPにて保守が楽になるか～
- 保守作業改善の基盤技術調査～保守の見積りと教育の現状～
- 保守のアウトソーシングを考える～保守業務におけるドキュメント整備の省力化～
- SMSGの考える保守とは

13年度(2003年度）・・・合宿地：長崎
- オープンシステムの保守～保守プロセスとメトリクス～
- 保守作業改善の基盤技術調査～保守の見積りと教育の現状～
- 保守のアウトソーシングを考える～分りやすい保守サービスの報告書とは～
- SMSGの考える保守とは

研究会のテーマ（6）

14年度（2004年度）・・・合宿地：盛岡
- オープンシステムの保守～ソフトウェア保守での作業プロセスの明確化と改善～
- 保守作業改善の基盤技術調査～GAMMA システムに関する調査～
- 保守のアウトソーシングを考える～保守の戦略的コストマネジメント～
- SMSGの考える保守とは

15年度(2005年度）・・・合宿地：徳島、CSKシステムズ
- オープンシステムの保守～保守しやすいソフトウェア～
- 保守作業改善の基盤技術調査～分散アプリケーションの進化およびメンテナンス支援に関する研究～
- 保守のアウトソーシングを考える～保守要員のローテーション～
- SMSGの考える保守とは

研究会のテーマ（7）

16年度（2006年度）・・・合宿地：別府
- 保守用ドキュメント・問題の抽出とアプローチ～問題把握および修正分析フェーズの検討を通して～
- 保守作業改善の基盤技術調査～分散アプリケーションの進化およびメンテナンス支援に関する研究～
- 保守のアウトソーシングを考える～保守スキル標準化による要員のローテーションの取り組み～
- SMSGの考える保守とは

17年度(2007年度）・・・合宿地：松山
- 保守用ドキュメントとは～隠れた保守ドキュメントの発掘に向けて～
- 分散アプリケーションの進化およびメンテナンス支援に関する研究
- 保守のアウトソーシングを考える～内部統制と要員のローテーションの活性化～
- SMSGの考える保守とは

研究会のテーマ（8）

18年度（2008年度）・・・合宿地：姫路、IKI
- 保守用ドキュメントとは～作業局面別 保守ドキュメントの考察～
- 分散アプリケーションの進化およびメンテナンス支援に関する研究」
- 保守のアウトソーシングを考える～SLAを支えるSE像～
- SERCの考える保守

19年度(2009年度）・・・合宿地：清水港
- 最低限必要な保守用ドキュメントを求め
- 保守作業改善の基盤技術調査　　保守のサービスレベル向上策
- SERCの考える保守

20年度(2010年度）・・・合宿地：桐生
- 現場の懊悩(オウノウ)をみんなのOh,yes!に
- 保守作業改善の基盤技術調査
- 障害を発生させない、被害を拡大させない対策とは
- SERCの考える保守

テーマから見える事

旬のテーマ
- Y2K、オープン、SLA等
- アジャイル系が少ない

要員、ドキュメント、プロセスに関するモノが多い

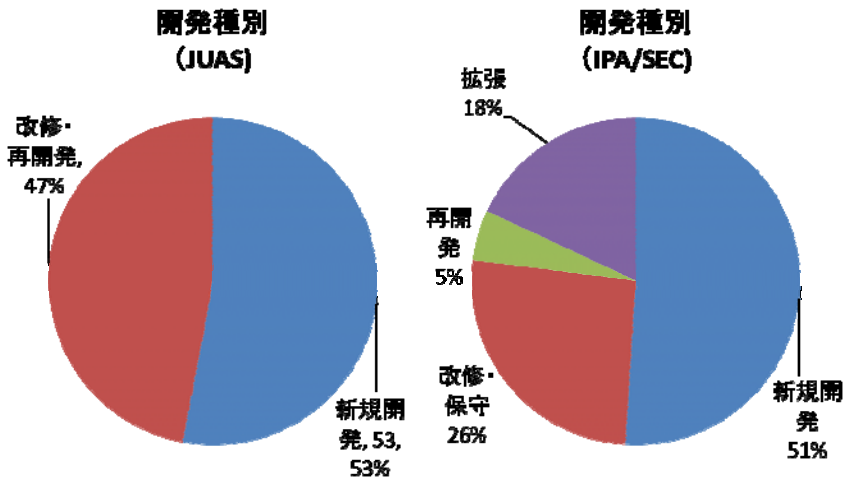
事例

Adobe Acrobat Document

今、20年前の問題はどうなってる？

- □信用できないドキュメント
  →変わっていない
- □特定の人しか知らない
  →変わっていない
- □保守のプロセスがバラバラ
  →保守プロセスが重視

現状の開発種別



ソフトウエアメインテナンス研究会
Ｃグループ
「１０年後の保守を考える」
第2回フォーラム

「ソフトウェア保守の自動化を目指して」

玄間　稔

「ソフトウェア保守の自動化を目指して」

１．１０年後は？？

①自動車は運転しなくてもいい？？

・「車の運転は人工知能を活用すれば事故を減らせる」

②ＩＴ業界の運用は自動化

・垂直統合型システム（ハードとソフト）

③ソフトウェア保守は？

・クラウド型や垂直統合型システムを前提にした保守

・個別対応型保守

10年後の保守を考える

10年後の保守作業環境について

ソフトウエアメインテナンス研究会
Cグループ　丸山陽一

近年、場所や時間を選ばず会社の情報にアクセスできるため、営業支援ツールや、
ペーパレス化に**タブレット端末やスマートフォンを導入する企業が増えています。**

一方、システム保守の現場では…

ユーザーが利用するデバイスが多様になっても、
従来と環境は大きくは変わっていません。

- PCの利用
- 作業場所が固定
- 紙の資料の利用
  (例えば、レビュー時に印刷する大量なドキュメントなど)

しかし今後はシステム保守の現場であっても

**タブレット端末や、スマートフォンを有効活用することにより、業務の効率化、
場所や時間を選ばない作業ができるようになるかもしれません。**

**そこで、今後のシステム保守業務においてデバイスフリーな環境がもたらす
影響を考えてみることにしました。**

デバイスフリー端末で保守作業ができるようになる？

- ✗ 物理キーボードが無くては作業が非効率
- ✗ 複数の画面を同時に見ることができない
- ✗ スマートフォンの小さな画面で何ができる？

作業場所、時間に縛られない保守作業ができる？

- ○ WEB会議などを活用することでコミュニケーション可能
- ✗ 夜間、休日でも障害対応が可能
- ○ 一線を退いた有識者とのコンタクトが容易となる

ペーパレス化の促進

- ○ レビュー資料の印刷が不要となる
- ○ いつでもドキュメントにアクセス可能

保守業務で取り扱っている機密情報の管理
保守作業契約の細分化
PCとの併用

有効活用できる場面と活用できない場面があり、これらの利用シーンと環境を整理することによって、システム保守作業に活用できるツールとなるのでは…

深堀を進めていく上でご意見等ございましたらよろしくお願いいたします。

私が考える10年後の保守　1/3　　Cグループ：伊藤 順一

- 10年後のコンピューターの普及予測
  - コンピューターやネットワークの進化や技術の発達により、企業・人・システムや物がどんどん繋がる時代になる。
    - (乗り物、家電、医療機器、計測機器、防犯装置、衣料品、文具 etc)
    - (ビジネスの変化、企業 M&Aによるシステム統合、EUCの普及 etc)
  - ハードウェアとソフトウェアはあらゆる物に普及し、それぞれが繋がるようになる
  - ソフトウェア保守開発は多様化・複雑化が進む。(なくなることはない)
    - 多様化1:メインフレーム、オフコン、サーバー、仮想化、クラウド　→　宇宙(ISS?)
    - 多様化2:専用端末、パソコン、携帯電話、スマートフォン、タブレットPC　→　チップ
    - 多様化3:キーボード、マウス、タッチペン、スキャナー(リーダー)　→　音声、脳信号
    - 多様化4:バーコード、QRコード、ICタグ　→　人間が見えないコード
    - 複雑化1:OS、ミドルウェア、部品、フレームワーク、ツール、言語　→　共通化
    - 複雑化2:DBMS(ネットワークDB、リレーショナルDB、ビックデータ　→　分散化
    - 複雑化3:手法(ウォーターフォール、プロトタイプ、アジャイル)　→　標準化
    - 複雑化4:プロセス(メーカー標準、国際規格、顧客プロセス)　→　各種規格の融合
- 例) 音声認識技術の発達で、音声による入出力が普及するも、キーボード・マウスによる入出力は、インターフェイスとして残る。

私が考える10年後の保守(伊藤)　2/3

- 予測による考察
  - ソフトウェア保守は10年後も変わらず存在する。但し、今よりも多様化・複雑化した状態での保守業務になっている。
    - オンサイト・オフサイト・自宅など拡散した環境に、地方や海外出張が増加
    - 日本人だけではなく海外の人と共同で保守開発を行なうようになる
    - 自律型サーバーが普及し、ソフトウェア保守者が運用を兼任する組織になる(ISOとITILの融合)
    - 影響分析ツールが登場し、繋がっているシステム全体の影響については、ある程度把握することは出来るようになる
    - プログラムジェネレーターが再登場し、言語依存がなくなる
    - ソフトウェアの部品化が進み、プログラム修正の量が減る
    - 古いソフトウェアの修正は手作業として、破棄されるまで続く
    - テスト自動化ツールが登場し、人手によるテスト負荷はある程度の下がる
    - 修正依頼では、是正保守は少なくなり、緊急保守と適応保守が多くなる
  - ソフトウェア保守技術者に必要なスキルは二極化する
    - 問題分析&修正分析では、システム全体を診れる人材の需要増
    - システム全体のテスト計画が立案でき、品質・進捗管理が出来る人材の需要増
    - ITと経営の一体化が進み、IT分野の専門化だけではなく、経営者とも会話出来る人材の需要増
    - ITSS、ETSS、UISSなどで定義された専門家

私が考える10年後の保守(伊藤)　3/3

- 若手ソフトウェア保守者に何を伝承・継承すべきか。
  - 私たちはエンジニア(技術者)なので、職人や芸能家のように、何十年もかけて伝承・継承することは出来ない(立場上はサラリーマン)
  - 断捨離TM(やましたひでこ)のように、絶って捨てて離れるわけにもいかない
  - ベテランの知識や技能を全てを教え込もうとすると、若手は新しい知識や技能を覚える余裕(伸びシロ)がなくなってしまう
- 伝承・継承すべきことは何か？どう進めるのか？
  - 何を伝承・継承したほうがよいのか？
    - システム・ソフトウェア進化の歴史
    - 顧客（組織）の文化
    - 保守・開発・運用プロセス
    - スキル(テクニカル、ヒューマン、コンセプチュアル)・エンジニアマインド
    - システム・ソフトウェア構成要素技術
    - 業務知識（ソフトウェア）
  - どうやって伝承・継承して行くのか？
    - 棚卸　→　取捨選択　→　OJTやOff-JTを使い分ける
- 何をどう伝承・継承するか議論中、報告書に間に合うかどうか…

10年後の保守

10年後の保守担当に求められるスキル

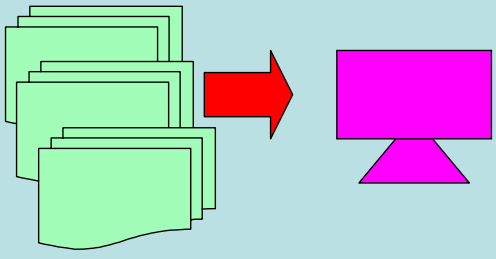
Cグループ　佐井　由美子

0. はじめに

一昔前と比べると、システムは単に自動化するための道具ではなく、より戦略的なツールとして要求されるようになっている

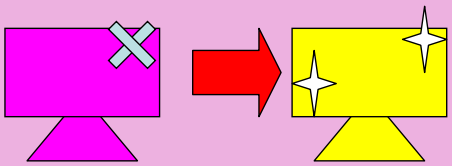
システム開発　と言えば・・・

一昔前までは

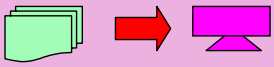


ペーパーレス化、業務の自動化
オフコンのWeb化
が主流

現在は



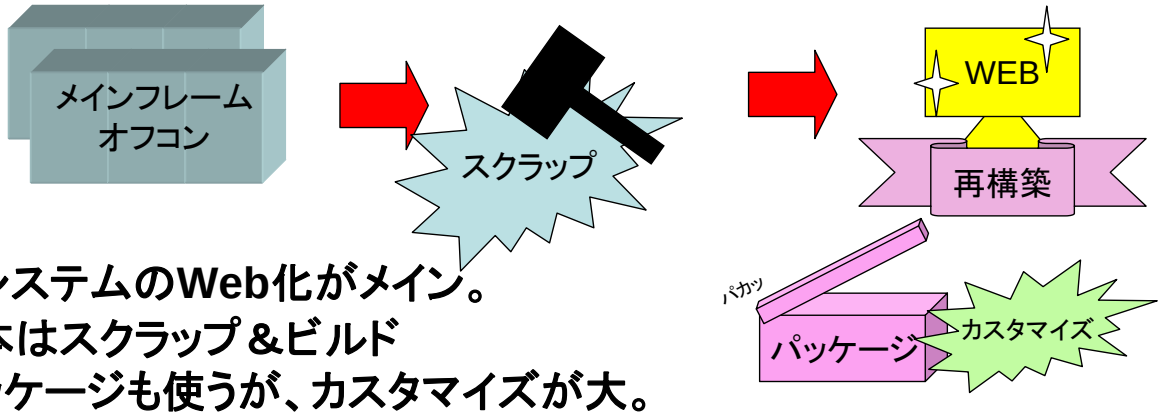
業務をよりよくするためのシステム
リプレース又は導入



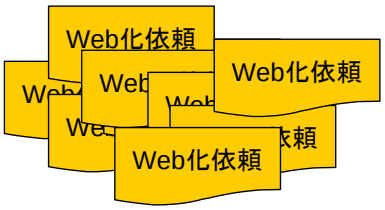
ペーパーレス化、業務の自動化は
飽和状態

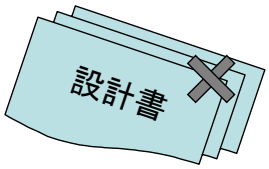
保守担当者はどのように変わっていくのか

1．10年前の保守（開発体系）

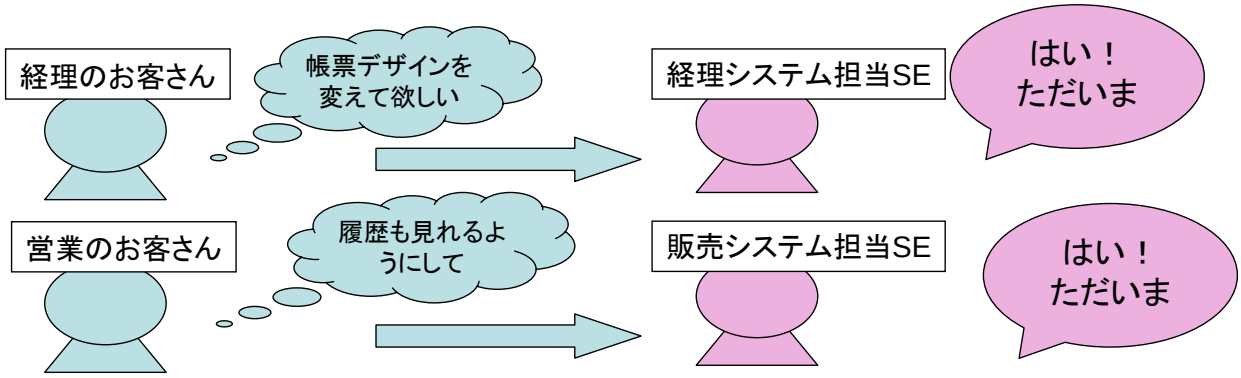


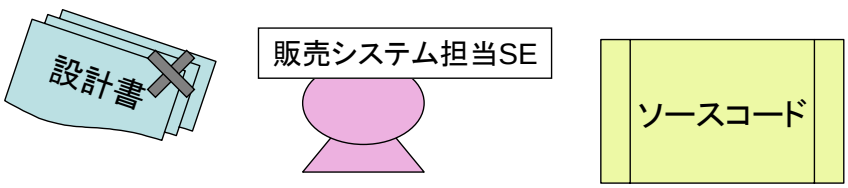
旧システムのWeb化がメイン。
基本はスクラップ＆ビルド
パッケージも使うが、カスタマイズが大。
次から次へと依頼があり、SE大忙し。
なので、ドキュメントは後回し。





1．10年前の保守
（保守担当に求められたスキル）



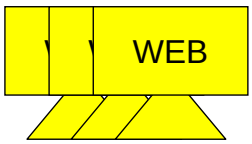


各システムごとに担当が割り当てられる
設計書は使い物にならないので、ソースコードから解析＆変更

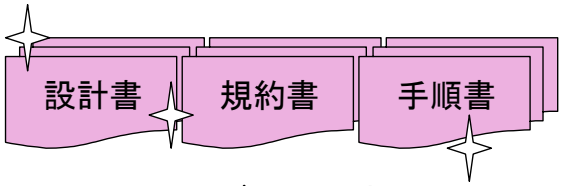
1．10年前の保守
（保守担当に求められたスキル）

- コーディングができること
- **担当するシステム**に関する業務知識があること（お客さんの要求が理解できるレベル）
- **担当するシステム**のロジック、ソースの場所が大体頭に入っていること

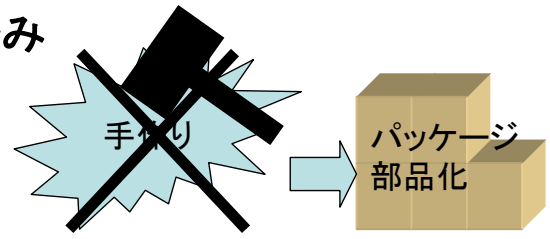
2．現在の保守（開発体系）



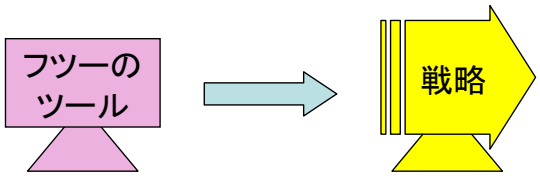
ほとんどがオフコンからリプレース済み



CMMなどでガッチリ整備

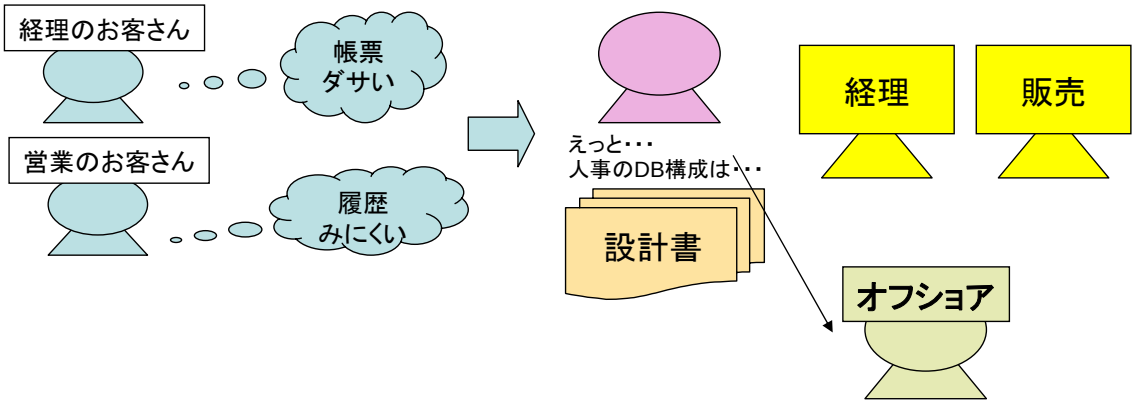


「作らないものづくり」推奨



お客様は単なる自動化道具ではなく、戦略的な武器として期待

2. 現在の保守
（保守担当に求めらるスキル）

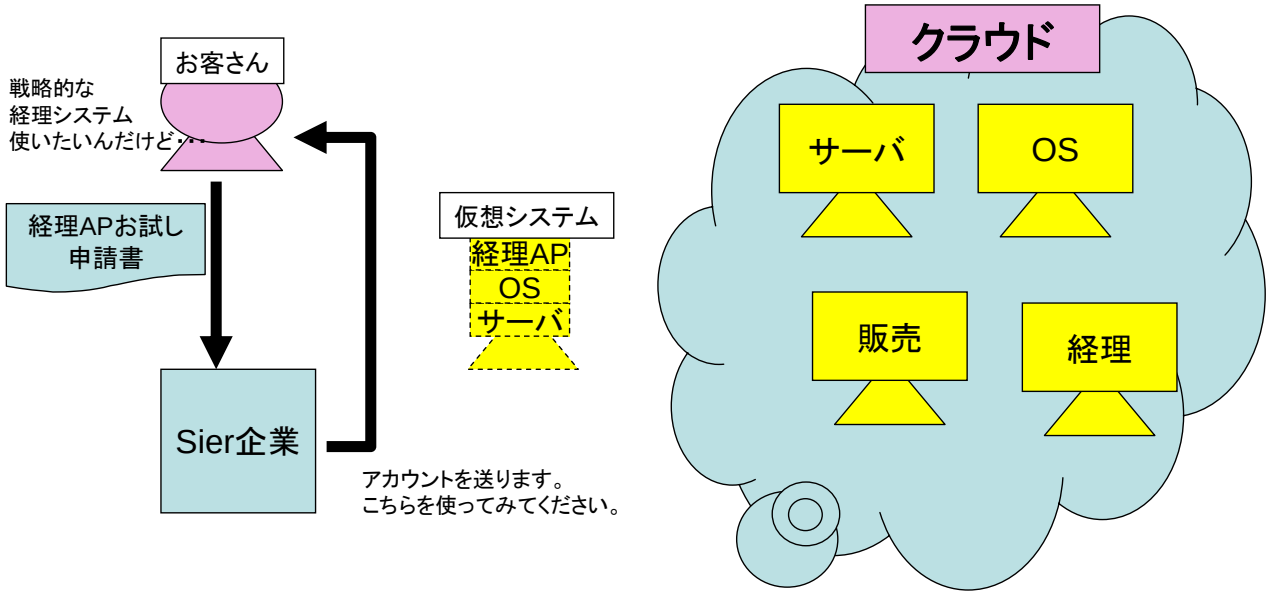


メンテされた設計書があるので、機能単位の「かけがえのない人」は不要に。
一人で複数機能の面倒を見るようになる
オフショア化やパッケージ使用のため、コミュニケーション力が必要

2. 現在の保守
（保守担当に求められるスキル）

- パッケージ機能を理解していること
- パッケージのパッチ情報を把握すること
- エンドユーザの話が理解できるレベルの業務知識があること
- **一人で複数機能を担当する場合もある**

3．10年後の未来予想図

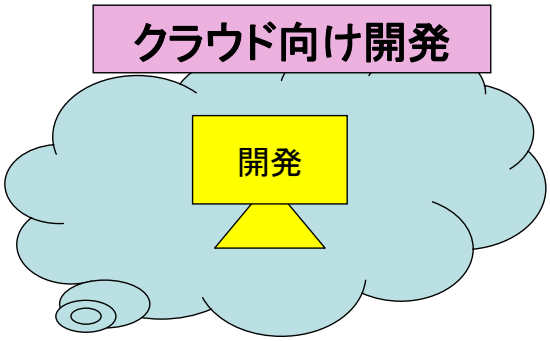


サーバ不要、インストール不要で申請のみでお試し版使用可能。
Sier企業も含めて、サーバを保有することもない
アプリ導入は開発ではなく「システム選び」がメイン。

4．10年後の保守（開発体系）

開発はクラウド向けシステム開発　か　システム選び　の2つ

コーディングはほぼ発生せず、
「設定」がメイン



クラウド上に部品が多数あるので
開発は少数

４．１０年後の保守
（保守担当に求めらるスキル）

・サーバ、OSも自前ではないので、
パッチあてなど基盤のメンテ不要
・部品の取替えか製品の乗り換えだけ
なので、コーディング不要
（一部のシステムは除く）
・システムの調査・検証がメイン
・業務知識を熟知し、クラウド上の幅広い
製品知識も必要。

５．２０２２年は

- 「要望にあわせて開発する」のではなく、「要望にあわせて取り替える」時代へ
- システムは「使い捨て」
- 作りこみがない＝開発と保守の線引きもない
- クラウド向けサービス開発以外は「保守　兼アプリケーション選択支援」
- 保守担当は人月￥２００万のコンサルタント同等のスキルを求められ、企業の将来を担う重要なポストに

保守開発のグローバル化

■10年後の保守開発

保守開発のグローバル化がいっそう進む
①. 海外へのビジネス展開
②. 国内SIビジネスにおける海外企業との協働、リソース活用　★主テーマ

【検討キーワード】
・オフショア開発
・システム開発の自動化（コーディングレス化）
・エンジニアが育つ環境（人財育成）

■10年後のSIサービス

・売上高と従業員数は横ばいで推移
・SIと運用が消える？
・大手SIerによる内製化（水平分業⇒垂直統合への回帰）
・オフショア（海外）でのリソース調達

保守開発のグローバル化に備える

■変化する環境への備え

①. 残るものへの強化、浸透
　・業務知識、上流工程、システム基盤
②. 進むものへのシフト、先行者利益
　・オフショア開発（海外）、自動化アプライアンス

■保守開発者への提言

・ひとりひとりの一芸強化（環境変化に負けないスキル）
・オフショア開発、海外リソースとの協働経験（ビジネス機会）
・自動化のフレームワーク理解（ソリューションの中にあるノウハウ・暗黙知）

「脅威」と怯まず、「機会」と捉える柔軟な思考で
明るい10年後を切り開いて行きましょう！

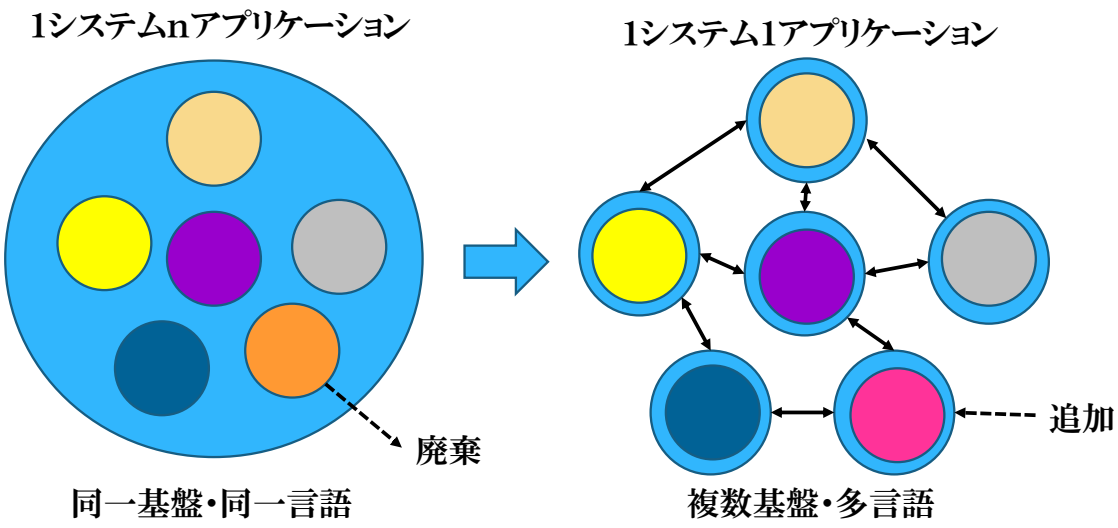
10年後の保守を考える

－ 10年後を見据えた人材育成 －

2012年 7月25日

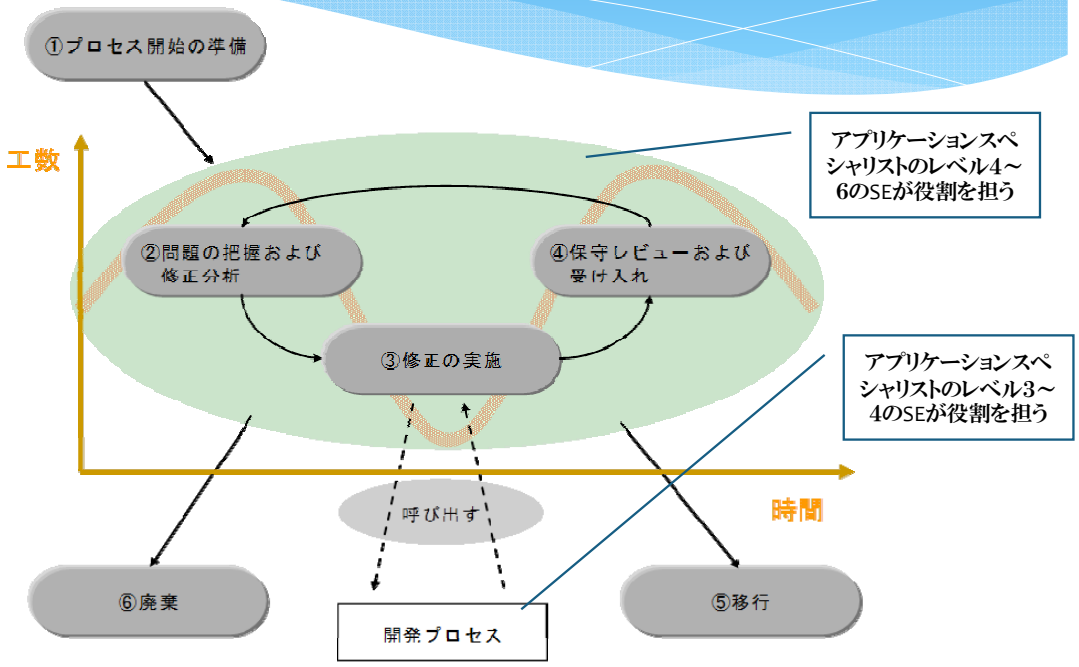
SERC－Cグループ 田中（創）

今後はシステムのコンポーネント化が加速



JIS X0161「保守プロセス」とSMSG「フタコブらくだ曲線」

保守の現状



ITスキル標準V3 キャリアフレームワーク

職種	マーケティング			セールス			コンサルタント		ITアーキテクト			プロジェクトマネジメント				ITスペシャリスト						アプリケーションスペシャリスト		ソフトウェアデベロップメント			カスタマサービス			ITサービスマネジメント				エデュケーション	
専門分野	マーケティングマネジメント	販売チャネル戦略	マーケットコミュニケーション	訪問型コンサルティングセールス	訪問型製品セールス	メディア利用型セールス	インダストリ	ビジネスファンクション	アプリケーションアーキテクチャ	インテグレーションアーキテクチャ	インフラストラクチャアーキテクチャ	システム開発	ITアウトソーシング	ネットワークサービス	ソフトウェア製品開発	プラットフォーム	ネットワーク	データベース	アプリケーション共通基盤	システム管理	セキュリティ	業務システム	業務パッケージ	基本ソフト	ミドルソフト	応用ソフト	ハードウェア	ソフトウェア	ファシリティマネジメント	運用管理	システム管理	オペレーション	サービスデスク	研修企画	インストラクション
レベル7																																		
レベル6																																		
レベル5																																		
レベル4																																		
レベル3																																		
レベル2																																		
レベル1																																		

クラウド化への対応が必須

IT投資局面と職種

IT投資の局面と活動領域 / 職種	経営戦略策定		戦略的情報化企画		開発		運用・保守	
	経営目標／ビジョン策定	ビジネス戦略策定	課題整理／分析（ビジネス/IT）	ソリューション設計（構造/ﾊﾟﾀｰﾝ）	コンポネント設計（ｼｽﾃﾑ/業務）	ソリューション構築（開発/構築）	ソリューション運用（ｼｽﾃﾑ/業務）	ソリューション保守（ｼｽﾃﾑ/業務）
セールス	目標/ビジョンの確認	ビジネス戦略の確認	ビジネス課題ソリューション提案					
コンサルタント	目標/ビジョンの提言	ビジネス戦略策定の助言	ソリューション策定のための助言	ソリューションの設計				
IT アーキテクト			ソリューションの枠組み策定	ソリューションアーキテクチャーの設計	コンポネントの設計	ソリューションの構築		
プロジェクトマネジメント			プロジェクト基本計画の策定	プロジェクトの管理/統制	プロジェクトの管理/統制	プロジェクトの管理/統制	プロジェクトの管理/統制	プロジェクトの管理/統制
IT スペシャリスト				システム構築計画の策定	システム・コンポネントの設計	システム・コンポネントの導入構築	システム・コンポネントの運用支援	システム・コンポネントの保守
アプリケーションスペシャリスト				ｱﾌﾟﾘｹｰｼｮﾝ開発計画の策定	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの設計	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの開発	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの運用支援	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの保守
カスタマサービス					導入計画の策定	ハードウェアソフトウェアの導入	ハードウェアソフトウェアの保守	ハードウェアソフトウェアの保守
ITサービスマネジメント						運用計画/運用管理の策定	システムの運用と管理	システムの運用と管理

重視

消滅

主たる活動局面　　従たる活動局面

10年後の保守を考えた時、必要な人材育成は？

* 課題
  * シニア保守要員の定年によるブラックボックス化加速
  * クラウド化による保守プロセスの変化
  * システム修正からシステム置換へ（コンポーネント化）
* 対策
  * 鳥瞰図によるシステム間インターフェースの把握・整理・継承
  * アプリケーション保守からシステム保守ができる人材育成（アプリケーションスペシャリスト→ITスペシャリスト）

２．作業グループ報告書

MORE REALLY, MORE USEFULLY!

～ソフトウエア保守ノウハウの改編～

グループメンバー	
アイエックス・ナレッジ(株)	**田中 一夫**
(株)SRA	**古石 ゆみ**
システム企画研修(株)	**上野 則男**
東芝ソリューション(株)	**沼田 恵助**
(株)バイトルヒクマ	**信国 智子**
	三村 千恵子
個人研究員	**塩谷 和範**
	中山 優紀
	福島 茂雄

2012 年 9 月吉日

目次

1.　本年度の活動 69

2.　昨年度との相違 70

2.1.　ノウハウ集拡充 70
2.2.　フォーラム 70
2.3.　現場に役立つ観点からの研究サブテーマ 70
2.4.　活動継続のための工夫 71

3.　変更結果の詳細 72

3.1.　個々のノウハウに対する課題の識別と対応方針、担当分け 72
3.2.　個々のノウハウの変更結果 72
3.2.1.　「見積もり・計画工程」に関するノウハウの見直し 72
3.2.2.　「要件定義工程」に関するノウハウの見直し 73
3.2.3.　「設計工程」に関するノウハウの見直し 74
3.2.4.　「本番環境での作業工程」に関するノウハウの見直し 74
3.2.5.　「不具合対応工程」に関するノウハウの見直し 75
3.2.6.　「引継ぎ工程」に関するノウハウの見直し 75
3.3.　まとめ 76

4.　次年度テーマの検討 77

1. 本年度の活動

2012/年度の活動として、2 回の合宿、7 回のミーティングによる計 9 回の活動を実施した。

表 2.1-1 本年度の活動一覧

#	名称	日程	場所	活動概要
1	キックオフ合宿	2011//12/9-10	三島	本年度の活動計画の立案
2	第 1 回定例会	2011//12/19	芝浦ふ頭	ノウハウ集の内容を「生々しく」直すよう決定
3	第 2 回定例会	2012//1/23	池袋	各ノウハウを「小芝居風※中山」の文体に修正
4	第 3 回定例会	2012//2/20	高田馬場	各ノウハウを「小芝居風※」の文体に修正
5	第 4 回定例会	2012//3/19	芝浦ふ頭	各ノウハウを「小芝居風※」の文体に修正
6	第 5 回定例会	2012//4/16	池袋	各ノウハウを「小芝居風※」の文体に修正
7	第 6 回定例会	2012//5/21	高田馬場	フォーラム実施について検討
9	第 7 回定例会	2012//7/17	芝浦ふ頭	報告書の方針決定、合宿について検討、次年度のテーマについて検討
10	グループ合宿	2012//8/24-25	葉山	報告書まとめ

※ 2.1 「ノウハウ集拡充」参照

2. 昨年度との相違

昨年度は、兼ねてから目標であったフォーラムの開催という手段でノウハウの拡充という目的が、達成できた。

その昨年度の結果を受けて、メンバーで今年度の方針を議論した。

2.1. ノウハウ集拡充

(1) 内容の拡充の継続

ノウハウに関し、SERC 内外に公開することで、新たなノウハウを拡充できないか、またより現場の役に立つノウハウ集を目指せないか、という意見が上がった。

(2) 内容に関して、有効な解決策を提案しているかの見直しとそれに関するリライト

追加の前に改めてリライトを検討してみた。解決策として妥当性がないという判断をされた対象について担当を決めて、解決策を見直した。その結果、解決策やノウハウは単純にカテゴライズされるものではなく、「生き物」であり、会話のような形が良いのではないかという案が出てきた。その会話形式の掛け合いを、回答者のキャラクターも織り交ぜて文章化することで、読み物としても、また寸劇にしても面白いのではないか、ということでそれを「小芝居」風と呼んだ。最終的にはなんらかの動画にしたい、という意欲はあったがそこまでたどり着けなかった。

2.2. フォーラム

(1) フォーラムを介した現場の声のすいあげ(インプット)

昨年度のフォーラムで、現場の声を沢山聞くことができ、多くのヒントが得られたが、問題点として出したいシチュエーションや、解決策として出したいノウハウの候補はなかなか抽出できなかった。それは得られたヒントからノウハウの形にするのが難しいということではないかと推測する。

それでもやはり、現場の声を聞き、ヒントを得る場として大切であると感じた。

(2) フォーラムを介した現場への還元(アウトプット・発信)

現場の役に立つ、そして現場とつながりを持ち、現場へ還元するために、フォーラムを実施していきたいが、どういったフォーラムをするべきか、という議論があった。

2.3. 現場に役立つ観点からの研究サブテーマ

2.2 で挙がった、フォーラムでのトピックとして、フォーラムを介した現場への還元にむけて、またサブテーマとして以下の内容が挙がった。

(1) ソースの複雑度について

ソースの複雑度が分岐の数や変数の数などで計れないかという議論になった。

(2) ソフトウエア保守における sustainability について

前年度の B グループのテーマ(sustainability)から、A グループのテーマである「現場」にリンクできないかという議論になった。

(3) サポート以外の保守についてのアピール

保守という言葉が世間的にフィールドサポートだと思われてしまうが、もっと保守という作業について認知度を高められないかという議論になった。

2.4. 活動継続のための工夫

定例会の年間計画を立て、メンバーが参加しやすい運営を検討したが、実際は各定例会に、メンバーの半数が参加できなかった。各メンバーの繁忙な業務との両立のために、移動と参加時間の負担を抑えた活動形態が必要ではないかという意見が出た。ホームページのリニューアルのタイミングで、SERC 内で導入された SNS をなるべく活用する形でノウハウ集拡充に関する作業を行った。

その議論から活動継続のための工夫も試みながら十分な活動に至ったのはノウハウ集拡充のみであった。

3. 変更結果の詳細

3.1. 個々のノウハウに対する課題の識別と対応方針、担当分け

上記 2.1 での検討結果に従い、個々ノウハウに対する課題を識別し、対応方針を検討、担当分けを行った。課題及び対応方針と担当者の一覧を表 3.1-1 に示す。なお、課題があっても対策に時間がかかりそうなものは、判定欄＝“保留”、担当＝“(未定)”とし、優先度を下げた。

表 3.1-2 個々のノウハウの課題と対応方針の一覧

No.	判定	課題及び対応方針	担当
1.3.1	保留	基本的にこのままでよい。	（未定）
1.3.2	保留	解決策として「見積もり精度を高めるためのドキュメント整備」を上げているが、理想論であり、すぐに対応できるものではない。	（未定）
1.3.3	対応	原因が具体的でなくイメージしにくい。原因を具体化し、即対応可能な解決策として見直す。	三村
1.3.4	対応	他社比較のベンチマークがあれば活用する	上野
1.3.5	対応	解決策が弱い。	塩谷
1.4.1	対応	設問が抽象的なので細かく分ける。	上野
1.4.2	対応	面白い設問だが、解決策の見直しが必要。	上野
1.4.3	対応	設問は提案を除けば OK。解決策見直し要。	上野
1.5.1	保留	事例として想定できない。	上野
1.5.2	保留	改善系。設問を補足する。	（未定）
1.6.1	対応	問題が大きすぎる。分割する。営業提案の話。	田中
1.6.2	対応	そもそもアクセスできることがまずい。回答見直し。	中山
1.7.1	対応	「利益分」ではなく「リスク」じゃないの？解決策見直し。	田中
1.7.2	対応	解決策が教科書的。	古石
1.8.1	対応	相談の②③が見えない。元のエクセルファイルを見ること。 プロジェクトの現状・メンバーの負荷、作業状況・対客情などは必要。規約等は見れば分かる。メンバーに聞けばいいことは引継ぎ不要。 相談内容がずれてる？＜引継ぎ時の・・・＞は解決策へもっていく。	福島

上記の方針と担当者に従い、各自で担当箇所を見直し修正案を提示し、グループ内でレビューを実施した。

3.2. 個々のノウハウの変更結果

個々のノウハウの変更結果を以下に述べる。

3.2.1. 「見積もり・計画工程」に関するノウハウの見直し

この工程のノウハウは、個人の見積もり能力を向上させる課題と、現場で顧客と対峙する局面で、

担当者としてどのような対処をすべきかという課題を取り上げている。ここでは、主に後者について、ノウハウとしてより具体的な対応策を提示できるように見直した。

保守の現場では、担当者が顧客に対して見積もりの交渉を行なわなければならない場面が多々想定される。回答に迅速さと正確さを求められ、精神的な圧力を受ける中、顧客が納得するロジックを立てて乗り切らなければならない。そのような担当者にとって役立つノウハウになるよう、具体的な受け答えや振る舞いをイメージできるような記述を心がけた。ノウハウ単位の改修概要を以下に示す。

図 3.2-1「見積もり・計画工程」に関するノウハウの改修概要

改修前		改修後のNo	改修概要	回答例有無
1.3.1	経験により見積もり能力に差が出るのですが	1.3.1	現状維持	
1.3.2	短い時間で行う"当初見積もり"の精度を高めたいのですが	1.3.2	現状維持	
1.3.3	過小見積もりのつもりで答えたら、概算見積もりにされてしまいました	1.3.3	問題を3つに分類し、分類した問題毎に解決策の見直し、回答例の追加	○
1.3.4	少なすぎる予算で、多すぎる要求内容を受託させられました	1.3.4	解決策の見直し、回答例の追加	○
1.3.5	直感に頼るのではなく根拠に基づく変更規模の見積もりをしたい	1.3.5	相談～解決策の見直し	

3.2.2. 「要件定義工程」に関するノウハウの見直し

この工程のノウハウは、要件の曖昧さに対応する課題、変更制御の課題、要件の理解に関する課題を取り上げている、

要件の曖昧さに対応する課題については、顧客との合意形成をどのように進めていくかを、具体的な思考や台詞を交えながら解説するよう、見直した。

変更制御の課題についても、注意すべき場面とその解釈、その場面に遭遇したときの具体的な振る舞い方をノウハウとして記述した。

要件の理解に関する課題は、保守担当者の永遠の悩みとも言える「顧客業務の理解」の課題に通じるものである。保守担当者は、完璧に業務を理解できていない状況でも、システム担当者として業務知識のポイントを押さえ、顧客と対等に議論を行い、必要に応じて提案を行なわなければならない。そのポイントの押さえ方、考え方を習得するための知見を、ノウハウとして記述した。ノウハウ単位の改修概要を以下に示す。

図 3.2-2「要件定義工程」に関するノウハウの改修概要

改修前		改修後のNo	改修概要	回答例有無
1.4.1	要求が曖昧で先に進んでいいか判りません	1.4.1	相談～解決策の見直し、回答例の追加	○
1.4.2	修正内容を具体化したところで顧客殿が当初の要件を変更してきます	1.4.2	解決策の見直し、回答例の追加	○
1.4.3	顧客殿業務の知識不足で、効果的な設計や提案ができません	1.4.3	解決策の見直し、回答例の追加	○

3.2.3. 「設計工程」に関するノウハウの見直し

この工程のノウハウは、「影響範囲の特定に有効なドキュメント」という課題に対する考察である。修正前のノウハウは「手順や文書の標準化」「新しい手法の導入」など、抽象的なセオリーのみの記述であった。ここでは、なぜ影響範囲の特定にドキュメントが必要かを解説し、CRUD 図の紹介を軸に解説を加えることにより、より具体的な「使える」ノウハウとした。ノウハウ単位の改修概要を以下に示す。

図 3.2-3「設計工程に関するノウハウの改修概要

改修前		改修後のNo	改修概要	回答例有無
1.5.1	影響範囲分析が楽に出来るドキュメントを教えてください。	1.5.1	問題の掘り下げ、回答例の追加	○
1.5.2	修正方法と意図が正しく伝わらなくて	1.5.2	現状維持	

3.2.4. 「本番環境での作業工程」に関するノウハウの見直し

この工程のノウハウは、本番リリースで担当者が抱える課題と、本番データへのアクセスに関わるコンプライアンスの課題を取り上げている。

前者のノウハウは、課題が多きすぎるため以下の3つに分解した。

- リリース手順がないシステムでのリプレース
- 引継ぎなしでリリース担当者に
- 資料なしのミドルウェアの入れ替え

後者のノウハウは、コンプライアンスを遵守しなかった場合の影響などを解説し、その重要性を解説するノウハウとした。ノウハウ単位の改修概要を以下に示す。

図 3.2-4「本番環境での作業工程」に関するノウハウの改修概要

改修前		改修後のNo	改修概要	回答例有無
1.6.1	リリース手順がないシステムでのリプレースを実施することになりました	1.6.1	オリジナルを生かし、解決策の見直し	
		1.6.2	タイトルを以下に変更し、全項目を見直し 「リリース担当者が、退職し引き継ぎも無い状態で、リリースを担当する事になりました」	
		1.6.3	タイトルを以下に変更し、課題及び解決策の細分化。 「ハードウェアの保守切れに伴い、ミドルウェアも入れ替える必要が出てきたが、ミドルウェアの資料が無いことに気付いた」	
1.6.2	本番データをテストアクセスしそうになりました	1.6.4	解決策の見直し、回答例の追加	○

3.2.5. 「不具合対応工程」に関するノウハウの見直し

この工程のノウハウは、不具合対応に至るまでの作業見積りの交渉に関する課題を取り上げている。顧客から見ると不具合対応は瑕疵であり、保守担当者には無料で実施させたい作業である。一方、保守担当者側は、たとえ不具合であっても、コストをかけて対応するため、無償では対応できない、という立場をとることになる。ノウハウ単位の改修概要を以下に示す。

図 3.2-5「不具合対応工程」に関するノウハウの改修概要

改修前		改修後のNo	改修概要	回答例有無
1.7.1	不具合を恒久対応するのに必要な工数を交渉したいのですが	1.7.1	解決策の見直し	
1.7.2	不具合対応の工数確保に苦労しています	1.7.2	相談～解決策の見直し	

3.2.6. 「引継ぎ工程」に関するノウハウの見直し

この工程のノウハウは、個人に蓄積されたノウハウの活用、及び、うまく引き継がれなかった場合を取り上げている。ノウハウ単位の改修概要を以下に示す。

図 3.2-6「引継ぎ工程」に関するノウハウの見直し

改修前		改修後のNo	改修概要	回答例有無
1.8.1	リーダー交代の引き継ぎが難航します	1.8.1	以下のサブタイトルを付加し、課題及び解決策の細分化。回答例の追加 「期間や時間に問題がある場合、どうすればよいのでしょうか？」	○
1.8.2		1.8.2	以下のサブタイトルを付加し、課題及び解決策の細分化。回答例の追加 「これって前任者と後任者のスキルの差なのかなぁ」	○
1.8.3		1.8.3	以下のサブタイトルを付加し、課題及び解決策の細分化。回答例の追加 「これって業務の範囲、作業量が問題なのだろうか」	○
1.8.4		1.8.4	以下のサブタイトルを付加し、課題及び解決策の細分化。回答例の追加 「これって引継ぐべき情報が問題なのだろうか」	○

3.3. まとめ

今年度は、より「現実的な」「生きた」ノウハウ集にするため、解決策の具体化または多様化を試み、適宜「ご参考　～回答例～」の欄を追加し「小芝居風」な表現で臨場感を持たせる工夫をした。保守の現場では、教科書的な解決策よりも、状況に応じた適切な解決策が求められると思われる。保守担当者は、問題に対峙する都度、多数の「解決策の引き出し」から、適切な解決策を選んで対応しなければならない。今回改訂したノウハウ集は、個々の担当者にとってその「解決策の引き出し」を増やすための一助となるものであると期待している。

4. 次年度テーマの検討

ここ 3 年ほど、保守の現場でのノウハウを集中的に研究してきて、付録のような成果ができた。今後は、このノウハウ集を公開し、皆さんの意見を頂き、年に1回程度の更新を行っていきたい。次年度からは、今回の研究で、ノウハウは大切だが、保守技術者の教育が重要である事を感じた。当研究会でも、2004 年度に保守技術者の教育カリキュラムを作成していたが、少々、古くなっており、技術的にも代わってきているので、この時代にあった保守カリキュラムを来年度以降研究します。

（付録）公開ノウハウ集 2011 年度版

1. 公開ノウハウ集 2011 年度版

1.1. ご使用になる前に

本ドキュメントは 2010 年に SERC-A グループが、これまでに関わってきた保守の現場での経験から困った場面を想定し、対処法としてノウハウ化したものをまとめたものです。本ノウハウ集は初版であり、まだ改善する余地がありますが、保守を行う前の留意点、行う際のヒントになるかもしれません。本ノウハウ集に書いてあることが“正解”ではなくあくまでヒントとして、ご自身の作業の環境や制約条件を考慮して応用してください。ご意見、追加ノウハウがあれば、SERC (serc-secretariat[at-mark]serc-j.jp)までお寄せください。ノウハウ集をよりよくするための参考にさせていただきます。

1.2. 使い方

一定の開発経験を積んだ方を対象としています。

プロジェクトや一つの作業に着手する前に読んで、リスクを洗い出すきっかけにしてください。

プロジェクト実行中に悩みが生じたとき、ヒントがあるかもしれません。

その他、何であれ参考にしていただければ幸いです。

1.3. 見積もり・計画工程

1.3.1. 経験により見積もり能力に差が出るのですが…(Yさん(49 歳)、プロジェクトリーダー)

相談

契約と直結している見積もりの場合、要求仕様や過去の変更結果、設計書、ソースリストなどを駆使して見積もりを実施しますが、やはり経験値によって見積もりにばらつきが発生します（見積もり項目がぬけているなど）。人によってばらつかない見積もりを実施するにはどうすればよいでしょうか。

問題

「見積り要素の見落とし」や「的確な数値を算出できない」などの事象が発生し、見積もりの精度が保てない。

原因

見積もり作業の手順や作業指針が明確になっていない。

あるいは、明確になっていたとしてもそれが守られていない。

経験が不足している担当者の場合これらの状況が独力では打破できず、結果的に品質のバラつきが発生してしまう。

解決策

経験値が少ない担当者に対しては、きちんと見積もりレビューを実施することが重要。

見積もりの詳細手順、見積もり用の様式、チェックリストを整備し、その規定に従って行われているかを有識者に確認させるといった方法が良い。

また、見積もりツール（ＦＰ換算）を用意するのも効果的である。

なお、次は考慮から抜けおちやすい項目なので注意する。

- データ整備
- マニュアル作成
- 管理コスト
- 設備費、導入費
- 保守用ドキュメントの作成、メンテナンス
- SLA を考慮した運用設計

組織的な活動としては、見積りレビューを実施するのが良い。

1.3.2. 短い時間で行う"当初見積もり"の精度を高めたいのですが…(Yさん(49 歳)、プロジェクトリーダー)

相談

見積もりのための”十分な調査”とは修正箇所の特定であり、修正箇所が特定できれば、ほとんどの保守案件が山を越えます。ほとんどの場合、短時間での″当初見積もり″が求められますが、 多くの場合、保守作業の進行においてその当初見積もり自体が制約となります。つまり保守作業中の計画変更は、当初見積もりから逸脱しない範囲で行われ、作業遅れが最後に発覚することが多々あります。限られた時間の中で当初見積もりの精度を高める必要がありますが、そのためにはどうするべきでしょうか。

問題

- 正確な見積もりに必要な調査時間が十分に与えられず、不正確な見積もりを作成してしまう。
- 与えられた工数以上に調査時間がかかる。

原因

時間が十分に与えられないのは問題だが、それが慢性化しているのであれば、製品保守としては調査工数がかかりすぎであることを意味する。

- 正確な見積りの勘所は、修正箇所の特定とそれを受けたテストの見積りである。
- それらの作業を考えたシステム構造やドキュメント構造になっていないため、工数が求められている以上にかかっている。

解決策

- 見積もり手順の整備にともなうドキュメントの整備
  これにより、”見積もり”作業の標準工数が算出できるようになる。これと、求められている見積もり作業工数を比較し、要求元に見積もり根拠を提示したり、より効率的な見積もり手順を開発したりできる。
- ツールとしては影響範囲特定ツール（高価だが効果あり）もある。

1.3.3. 過小見積もりのつもりで答えたら、概算見積もりにされてしまいました…(Fさん(36 歳)、チームリーダー)

相談

過小見積もり（概算）のつもりで回答して、公式見積もり（詳細）として扱われてしまい苦労し、挙句の果てその公式見積もりが大きく外れていると非難されることが多いです。

問題

- 過小に見積もってしまう（直感・検討漏れ）
- 正しい手順を踏まず回答している（回答すべきではない人が回答している）
- 概算なのか詳細なのか意識していないが、概算で出したつもりが相手に詳細見積もりとして受け取られる

原因

- 客先から電話口で見積回答を迫られたことで、「すぐに答えを出さなければ」「あまり余裕のある見積では、受注が取れないのでは」というような焦りがあったため
- ざっくりとはいえ、SE が回答した見積なんだからそう大きくは外さないだろう、という期待等から、正式見積を怠ったため
- 暫定見積者の見積スキル不足

解決策

- 見積が外れたときに非難されるのが嫌なら、焦って回答しない。
- 正規の見積を後日行う、という手順を遵守する。
- 見積スキルの向上を図る(ＰＳＰなどで自分の能力を測っとけ！)

ご参考～回答例～

相談者「営業から『今お客さんのとこなんだけど、変更するとしたらいつまでに出来るか、ざっくりでいいから教えて。受注かかってるんだよ！』なんて電話がかかってくるんです。

『受注取らなきゃ！』なんてバイアスがかかると、その場で「うーん。一ヶ月」なんて答えてしまうわけですよ…。よくしちゃってます♪」

回答者Ａ:「それって、回答手順の問題じゃないんですか。急ぎとはいえ、一個人の直感で見積を即答したり、正規の見積回答の手順に従わなかったっていうのは…やばいですねぇ。作業スコープを決定し、影響調査を実施してから、後日正式見積を行う手順を遵守するのがセオリーですから、その場合、『影響を調べるので、いついつ回答します。その上で後ほど正式に見積依頼を出してください。』というのが正しいのではないでしょうか。」

回答者Ｂ:「見積の回答の仕方として、段取りを考える方向に持っていくっていうのがありますよね。『お客さんは何がしたいんですか？なるほど。ではＡとかＢとかも出来ますよね。決まってないなら、提案しましょう。いつします？』てな感じです。」

回答者Ｃ：「やはり、その場ですぐに回答しなければならない場合は、直感的な見積よりかなり余裕のある期間で回答する、というのはありますね。それでは時間がかかりすぎると言われた場合は、交渉にもっていくべきでしょう。」

回答者Ｄ：「見積もりのズレ、と言う意味では、作業スコープを決める段取りの問題があるでしょうね。要求分析の要否、打ち合わせの要否、納期（顧客のインストール可能時期の考慮）、予算などを考慮しなかったことが効いてくるんですね。その意識から『要求分析や打ち合わせは必要でしょうか？どこまで話ついてます？』と訊くのもいいでしょう。見積が実績と大きく外れてしまう、っていうのはやはり個人の見積スキルを上げる、出てきた見積もりに組織判断でリスクやスキル、不明点を考慮した工数を乗せるということも考えた方がいいですね。」

.結論：「首絞まる　八方美人に　『ちょっと待った！』」

1.3.4. 少なすぎる予算で、多すぎる要求内容を受託させられました…(Sさん(34 歳)、開発担当)

相談

顧客殿要求が膨大であるにもかかわらず、予算有りきの案件であることが多いです。営業からの圧力が大きい場合が大半で、妥当な見積もり金額を開発から出して受託することはほとんどありません。

問題

予算と要求が折り合わない。

原因

- 開発する要求事項と予算のトレードオフを基にした予算交渉ができていない。
- 予算ありきであれば、要求事項を削るべきなのかもしれない。それらを含めた合意がされていない。

解決策

1. 根本的な改善方法（今後に備える対策）

見積り時に営業に対する開発側の意見を反映できるような見積りの責任体制を明確にする。営業的観点から政策的に不利な条件で受注するということであれば、開発予算（計画）はそれを加味した内容にして、開発側の責任ではないようにする制度を設ける。

例：１０００万円で受注するが、開発側は１５００万円で開発する予算とし、差額の５００万円は会社の負担とする。

2. 正当な見積り手順

要求事項、予算、求められる品質を明示した上で

- 顧客側の主張の根拠を確認し、開発側の根拠を示す。
  例：他社比較などのベンチマークを提示する。
- 顧客の利用部門などから、具体的な要求を把握する努力をおこなう。
  具体的な目的や予算・日程の内容を話すと、要求の優先順位や納入日程など、現実的なスケジュール交渉が可能となる。
- その上で、予算・納期条件が絶対であるなら、その範囲で収まるように実現する機能を絞る交渉を行う。
- 実現機能が必須であるなら、それが実現できる体制や納期・費用等を検討し、交渉を行う。
- 決着は、交渉力。政治力、経営的判断で決定される。決定したらその実現に最善の努力を行う。

ご参考～回答例～

相談者「予算と要求が折り合わないんです…こんなバカな話、ありますか…！！私は普通の、ノーマルな仕事がしたいだけなのに…！！！」

回答者 A「それは、開発する要求事項と予算のトレードオフを基にした予算交渉ができていないということですよね。予算ありきなら、要求事項を削ることだってある、それらを含めた『合意』ができていないことが問題です！

正当な見積り手順を示すと、次のようなステップになります。

⑴お客様側の『要求事項、予算、求められる品質』と、開発側の認識する『要件を明示化』しよう！そして主張の『根拠』を相互に明確にしよう！！

例：他社比較などのベンチマークを提示する。

お客様と開発の対話による認識合わせが、まず第一です！！！

⑵お客様の利用部門などから、具体的な要求を把握する努力をしよう！

具体的な目的や予算・日程について話すと、

・要求の優先順位

・納入日程

など、現実的なスケジュール交渉が可能となります！！

⑶その上で、

・予算・納期条件が絶対なら

→その範囲で収まるように実現する機能を絞る交渉しよう！！

・実現機能が必須なら

→実現できる体制や納期・費用等を検討し、交渉しよう！！

ってことだね。対話の機会をつくらなきゃ、同じ土俵で交渉できないですよ。」

回答者 B「今後に備える対策として、根本的な改善方法も考えられますよ。見積り時に営業に対して開発側の意見を反映できるような、『見積り責任体制』というのを明確にしたらいいんですよ。

営業的観点から政策的に不利な条件で受注する、ならば、開発予算（計画）に関しては営業の責任で負担してもらうんです。

例：１０００万円で受注するが、開発側は１５００万円で開発する予算とし、差額の５００万円は会社の営業の負担とする。」

結論：社内外！ネゴって笑え！！『対話』王！！！

1.3.5. 直感に頼るのではなく根拠に基づく変更規模の見積もりをしたい…(Oさん(39 歳)、プロジェクトリーダー補佐)

相談
変更作業の見積り時に顧客要求が明示できない場合、客観的な根拠なしに担当者の直観で変更規模を見積る傾向にあり、大きな工数変動リスクになっている。工数増加防止＆納期厳守のためには、着手前に顧客要求の分析を行うことが不可欠であることを、顧客に理解してもらい正確な見積もりを行いたい。
問題
顧客要求がハッキリしないにもかかわらず、独りよがりで感覚的な見積りが行われると、作業開始後に新たな要求が発生しがちなため、見積工数を上回る実作業工数になってしまう傾向にある。
原因
見積り時に顧客要求が明示できないのであれば、顧客要求を引き出す作業も見積り作業に含まれ、これが工数増加になる。さらに、顧客要求の引き出しと分析が不十分な場合、作業中に新たな要求が発生することがあり、さらなる工数増加になる。 根本的には、顧客の変更意図を十分に把握できていないことが原因として考えられる。
解決策
まずは、作業着手前の顧客とのコミュニケーションを最優先とし、変更意図を把握することが肝要。その中で、代替提案も示すことができればベスト。 また、見積りのためには要求分析が必須であることを納得していただき、必要な作業として明示し顧客の承認を得る。 顧客は具体的な成果物のイメージを持っているかもしれないし、それも含めて提案を期待しているのかもしれない。あるいはやりたいことはきわめて単純なのかもしれない。 まずは顧客とのコミュニケーションによって認識あわせを行う必要がある。 直観も顧客要求引き出しのためには重要な糸口となる。直観に基づく仮定を顧客にぶつけることにより背後にある顧客の意図を探ることができるかもしれないし、これをきっかけに明示されていない顧客要求を引き出せるかもしれない。あるいは、直観の誤りが未然に分かることもある。 仮定：顧客要求引き出し→要求分析→システム化要件確定→顧客承認→変更作業開始

1.4. 要件定義工程

1.4.1. 要求が曖昧で先に進んでいいか判りません…(Oさん(39 歳)、プロジェクトリーダー補佐)

相談
顧客殿からの変更要求が曖昧で抽象度が高かったので、具体的な中身を決めてもらおうとしていますが、顧客殿からの要件がまとまらず先に進まないでいます。もうそろそろ設計を開始しないと、工程が遅延してしまいますが、今顧客殿から出ている話をベースに設計に進んでもよいでしょうか。このまま設計・実装に進んで、要件をひっくり返されるリスクが心配です。
問題
• 変更要求が曖昧である。 • 要件が確定できない。 • 顧客の真の要求を満たせない。
原因
• 要求事項について顧客と合意する、という工程が不明確 • 顧客の真の要件を引き出せない
解決策
相談者への回答としては、要件があいまいなまま先へ進んではいけない、ということである。 要件があいまいなまま先へ進んで成功した例は皆無と言ってよい。 先ず、「いつまでに要件を確定させないと納期が維持できません」ということを、顧客側に明示する。 開発側が持っているカード（どういう修正が考えられるか等）を都度提示し、顧客に要件をすべて吐き出していただくよう努める。 顧客は開発側が求めている具体化のイメージを持てていないのかもしれない。顧客は具体的な成果物のイメージを持っているかもしれないし、それも含めて提案を期待しているのかもしれない。あるいはやりたいことはきわめて単純なのかもしれない。まずはこの認識あわせを行う必要がある。（できるだけ設計工程に進む前までに済ませておきたい）要件獲得の手法や要件定義の手法がいくつかで出ている。 例：ＭＩＮＤ－ＳＡ　（http://www.newspt.co.jp/contents/manual/index.html） その中に、顧客が本当に望む要求を引き出す有効な方法がある。 それは、システム要件を必要としている業務上の「ねらい」を明らかにすることである。 誰が何のために、この要件を必要としているのかを確認する。 「ねらい」は、「早い、うまい、安い、（ＱＣＤ）」で確認するとよい。 「業務を早く処理する必要はありますか、その業務は何ですか、どの程度早くする必要があるのですか。それは誰が望んでいるのですか」 「業務のレベルアップを行う必要はありますか、その業務は何ですか、どの程度レベルアッ

プする必要があるのですか、それは誰が望んでいるのですか」

「業務の省力化やコストダウンは必要ですか、その業務は何ですか、どの程度省力化やコストダウンする必要があるのですか、それは誰が望んでいるのですか」

そうして、要求がはっきりしなければ、望んでいる人の意見を確認するようにす。る。

「ねらい」が固まれば、「ねらい」を実現するためにはシステムでは何ができなければならないかの検討を行うことができる。

システム要件の起点は業務上の「ねらい」なのである。

·そして、決まっていない要件は何か。それによって生ずるリスクを明確にし、決めていく方法を顧客と話し合う。

·要求分析自体を期待されているなら、必要な作業として明示し、まずはこの見積もりについて話す。

·窓口になっている顧客が要件を出しきれないのは、窓口の担当では整理できないのかもしれない。他の部門や他の担当、上司の意向があるのかもしれない。その場合は、この案件のステークホルダを確認し、その方々の検討への参画をお願いすることが必要である。

ご参考～回答例～

相談者「変更要求が曖昧だったり、要件が確定できなかったりします。お客様の真の要求を満たせないのが、後から分るとえらい手戻りになりそうで…」

回答者 A「要求事項についてお客様と合意する、という工程が不明確なんじゃないかと思います。まず第一に要件があいまいなまま先へ進んではいけません。要件があいまいなまま先へ進んで成功した例は皆無と言ってよいでしょう。「いつまでに要件を確定させないと納期が維持できません」ということを、お客様に明示するべきですね(要件確定納期の明示)。」

回答者 B「お客様は具体的な成果物のイメージを持っているかもしれないし、持っておらず、それも含めて提案を期待しているかもしれない。明確にすると単純な要件なのかもしれない。認識合わせが重要です！開発側が持っているカード（修正方法案など等）を都度提示し、お客様に要件をすべて吐き出していただくという方法もあります(具体案を示すことによる潜在要件の洗い出し)。」

回答者A「こんなものもありますよ。できるだけ設計工程に進む前までに済ませておきたい）要件獲得の手法や要件定義の手法がいくつかで出てます。

例：ＭＩＮＤ－ＳＡ　（http://www.newspt.co.jp/contents/manual/index.html）

お客様の本当に望む要求を引き出すのに有効なのは、システム要件を必要とする業務上の「ねらい」の明確化です。誰が、何のために、この要件を必要としているのかを確認してください。」

回答者B「また、「ねらい」は、「早い、うまい、安い、（ＱＣＤ）」で確認することができます。

・「業務を早く処理する必要はありますか、何の業務を、どの程度の早さを、誰が望んでいるのですか」

・「業務のレベルアップを行う必要はありますか、何の業務を、どの程度のレベルアップを、誰が望んでいるのですか」

・「業務の省力化やコストダウンは必要ですか、何の業務を、どの程度の省力、コストダウンを、誰が望んでいるのですか」

要求がはっきりしなければ、「望んでいる人」の意見を確認するようにしてください。」

回答者 A「「ねらい」が固まったら、実現のための必須事項の検討ができます。つまりシステム要件の起点は業務上の「ねらい」だと言えます。

決まっていない要件はそれによるリスクを明確にし、決めていく方法をお客様と話し合い、リスク管理をする必要があります。」

回答者 B「要求分析自体を期待されているなら、必要な作業として明示し、まずはこの見積もりについて話す必要があります。」

回答者 A「窓口になっているお客様自身では要件を出しきれない、整理できない場合もあります。他の部門や担当、上司の意向があるの場合は、この案件のステークホルダを確認し、検討への参画をお願いすることが必要です。 」

結論：早・うま・安　誰が何を　どのくらい

1.4.2. 修正内容を具体化したところで顧客殿が当初の要件を変更してきます…(Fさん(36 歳)、チームリーダー)

相談
顧客殿が、こちらの修正内容が分ってくると、当初「やりたいこと」と言っていたことと違うことを言い出します。どうしたらよいでしょうか。
問題
・ 「違う」ことを取り入れると、当初の見積もりや計画を超過してしまい、場合によっては、希望納期に間に合わないこともでてくる。 ・ あるいは、自分の工数を予定以上にとられてしまい、他の案件に影響を与える。
原因
このような問題が発生した原因は、検討開始時に顧客も開発者も「やりたいこと」を引き出し切れていなかったことである。その問題が発生する原因は二つ考えられる。 ・ 顧客殿は「やりたいこと」が明確だったが、当方がそれを引き出し切れなかった、という場合 ・ 顧客殿も当初は「やりたいこと」が必ずしも明確てなかった場合
解決策
【本来の解決策】 ・一つ目の原因に対する対策は、検討開始時に、受けた「あいまいな」要求に対して、「そのご要求を実現するとすれば、システム的な対応はこうすればよいのですか」とどんどん具体化して提示すればよかったのである。それを示されれば、「そうだ」とか「違う、こうだ」とか言っていただけ、その時点で要件が固まったはずである。 ・二つ目の原因に対する対策は、一つ目の対策のようにはいかない。相手がまだ自分のやりたいことが明確になっていないからである。この場合は、「これをなさりたい目的は何ですか？何のためにこれをなさりたいのですか？業務のどんな問題を解決または改善なさりたいのですか？」という質問から始める。そうすると、一緒に考えながら「やりたいこと」を具体化していくことができる。 ・一つ目の原因に対する場合でも、この質問を投げかけるところから始めてもよい。 【現時点での解決策】 ・すでに、「違う」ことが出始めている、現時点での対応法は、以下のようになる。 ・先ずは遅ればせながら、現時点で要件を確定させることである。そうしないとまだまだ「違う」要求が続出する可能性がある。 ・そのために、上記の「これをなさりたい目的は何ですか？何のためにこれをなさりたいのですか？業務のどんな問題を解決または改善なさりたいのですか？」という質問から始めて、要求を引き出し要件を確定させる。 ・その上で、その案で実現する場合の見積りを行う。その見積りが当初案の場合とほぼ同様で

あり、納期・必要費用に影響を与えないのであれば、その案で実行する。

・当初見積りの線ではいかず、納期・必要費用に影響を与える場合は、以下のように対応する。

要求変更・仕様変更であることを相手に説明し、納期・費用変更を要求する。認められればよし。認められない場合は、実現機能を絞って納期・費用の範囲内で実施することを提案する。

・これから先は「交渉力」である。交渉力が弱ければ膨らんだ要求を当初条件で実現しなければならないかもしれない。

ご参考～回答例～

相談者「言われた「違う」ことを取り入れると、当初の見積もりや計画を超過したり、納期に間に合わないこともあるんです…予定以上に工数をとられて他の案件に影響が出たりもします。」

回答者 A「検討開始時にお客様も開発者も「やりたいこと」を引き出し切れていなかったんでしょうね。そういう場合、開発者側が引き出しきれなかった場合と、お客様自身にも解がなかった場合があります。

開発者が引き出しきれなかった場合は、検討開始時の「あいまいな」要求に対して、「ではシステム的な対応はこうですか」とどんどん具体化して提示すればよかったのだといえます。

具体化すれば「そうだ」とか「違う、こうだ」と反応をいただけ、要件を固められるはずです。

お客様の中に解がない場合は、「ねらい」を見極めることですね。「これをなさりたい目的は何ですか？何のために？業務のどんな問題を解決または改善なさりたいのですか？」という質問から始めましょう。

一緒に考えながら「やりたいこと＝要件」を具体化していくことができると思います。この方法は、開発者が分っていない、前者の場合にも利用できます。」

回答者 B「すでに、「違う」ことが出始めている場合は、まず現時点での要件の確定から始めましょう。まだまだ「違う」要求が続出する可能性があります。

そのためには同様に「ねらい」の見極めを行いましょう。その上で、実現するための見積りを行い、当初案の場合とほぼ同様の納期・必要費用であれば、その案で実行しましょう。

当初見積りの線でいかない場合は、要求変更・仕様変更であることを相手に説明し、納期・費用変更を要求しましょう。認められない場合は、実現機能を絞って納期・費用の範囲内で実施することを提案しましょう。

この先は「交渉力」です。負けたら「膨らんだ要求を当初見積もりで行う」という無茶が待ってます…！」

結論：「分った」が　つもりだったと　悟る時…(恐怖の始まり)

1.4.3. 顧客殿業務の知識不足で、効果的な設計や提案ができません…(Oさん(39 歳)、プロジェクトリーダー補佐)

相談

顧客殿業務に対する知見が不足しており、実際の顧客殿業務に即した保守作業や提案ができません。設計ミス、提案ミスを呼び込んでしまいます。

問題

- 知識が不足した状態で保守作業に携わっていること
- 提案作業が行えず、顧客から軽く見られてしまう恐れがあること

原因

- 作業に必要な知識が明確になっていない。顧客業務がわからない。
- ということもあるが、いわゆる業務知識は奥行きが深く、単に「知る」ということを目的にすると、きりがないこととなる。最低限の業務知識を補うのは、相手から必要な情報を引き出す能力である。その能力が不足していることも相談者の問題であると想定される。

解決策

【前提条件】

その業界の用語などがまったくわからないと、相手と会話をすることもできない。最低限度の業務の理解をしておく必要はある。生産、物流、在庫、営業、販売の仕組みはどうなっているのか、である。

業界の解説書が数社から出ているので、該当の解説書があれば勉強できる。

（産学社）産業と会社研究シリーズ

（日経文庫）業界研究シリーズ

（日本実業出版社）業界の最新常識シリーズ

【業務プロセスの理解】

前提条件の最低知識は理解している上で、対象業務のプロセスを理解する。

業務は、開発、購買、生産、物流、営業などの業務が、相互に関係しながら階層構造を形成している。その階層構造を理解する。階層構造の最下層は業務プロセスフローである。

業務の体系・構造を知る業務プロセスモデルとしては、ＡＰＱＣ（米国生産性品質センター）で作成しているモデルが有名で、英語版だがダウンロード無料利用ができる。

システム企画研修社が提供しているＰＤＲ（標準業務プロセスモデル）は、有償だが具体性と詳細度が高い。

各業務がどのように実施されているのかは、顧客が作成している業務マニュアル等で勉強する。

特定の案件のニーズと関係なしに、業務プロセス等の整理をすることは、労多くして功少ないのでお勧めできない。

【業務ニーズの引き出しまたは理解】

業務プロセスの基本的な理解をした上で、本件の具体的なニーズの引き出しや理解は、顧客との対話によって実現する。

「今回の対象業務はこれですね」と、対象業務のハコを示す。

「これについて、どういうご要望があるのでしょうか」

「現状ではどうされているのですか」

「それを改善なさりたい目的はどういうことなのでしょうか」という風に聞いていく。

この際、理解できない用語等が出てきたら、あいまいにしないで以下のように質問する。

「勉強不足でたいへん申し訳ないのですが、〇〇というのはどういうことでしょうか」

特別ご縁の深い顧客の業務用語理解のために、業務に即した資格試験の獲得、などもできればチャレンジするとよい。

【相談事項を今すぐに実現したいなら】

【前提条件】をクリアできているのであれば、早急に【業務プロセスの理解】にチャレンジし、【業務ニーズの引き出しまたは理解】を実践してみてください。

それができないなら、しばらくできる人の力を借りて前進しましょう。

ご参考～回答例～.

相談者「知識が不足した状態で保守作業に携わっているんだとは思います…提案作業が行えず、顧客から軽く見られてしまうのも、耐え難いです…」

回答者 A「作業に必要な知識が明確になっていない、ってことはないですかねぇ。顧客業務がわからない、ということもありますが、

いわゆる業務知識は奥行きが深いし、本当の意味で「知る」っていうのは大仕事ですよ。

「最低限の業務知識」を補いましょう。それには相手から必要な情報を引き出すべきですね。」

相談者「聞くことで、「最低限の知識」を入手するんですね。」

回答者 B「その業界の用語などがまったくわからないと、相手と会話をすることができないですから、「最低限の業務」理解をしておく必要はありますね。

生産、物流、在庫、営業、販売の仕組みなどです。業界の解説書が数社から出ているので、それで勉強できますよ。

（産学社）産業と会社研究シリーズ

（日経文庫）業界研究シリーズ

（日本実業出版社）業界の最新常識シリーズ」

回答者 A「「最低限の業務」は理解している上で、対象業務のプロセスを理解するべきですね。

業務は、開発、購買、生産、物流、営業などの業務が、相互に関係しながら階層構造を形成しています。その構造を理解するんです。
階層構造の最下層は業務プロセスフローです。
・ＡＰＱＣ（米国生産性品質センター）で作成しているモデル(業務の体系・構造を知る業務プロセスモデル。英語版だがダウンロード無料利用ができる)
・ＰＤＲ（標準業務プロセスモデル）(システム企画研修社が提供しており有償だが具体性と詳細度が高い)
なども活用してみるとよいですね。各業務がどのように実施されているのかは、顧客が作成している業務マニュアル等で勉強しましょう。
特定の案件のニーズと関係なしに、業務プロセス等の整理をすることは、労多くして功少ないのでお勧めできません。」
回答者 A「具体的なニーズの引き出しや理解は、業務プロセスの基本的な理解をした上でのお客様との対話によって実現します。
「今回の対象業務はこれですね」と、対象業務のハコを示し、(対象)
「これについて、どういうご要望があるのでしょうか」(要望)
「現状ではどうされているのですか」(現状把握)
「それを改善なさりたい目的はどういうことなのでしょうか」(目的・根拠)という風に聞いていきましょう。
その際、理解できない用語等が出てきたら、あいまいにしないで
「勉強不足でたいへん申し訳ないのですが、〇〇というのはどういうことでしょうか」と質問しましょう。」
回答者 B 「特別ご縁の深いお客様なら、業務用語理解のために業務に即した資格試験の獲得などもチャレンジするといいかもですね！」
回答者 A「最速コースは「最低限の業務」理解ができていれば、業務プロセスを理解する段階に進んで、業務ニーズの引き出しまたは理解に挑むことですね。
それができないなら、しばらく「できる人の力を借りて前進」するのも手です。」
結論：「質問を　するにもまずは　勉強です」(ツライことだからこそ、お金がもらえるってもんです…)

1.5. 設計工程

1.5.1. 影響範囲分析が楽に出来るドキュメントを教えてください。 (Yさん(49 歳)、夢見るプロジェクトリーダー)

相談
修正箇所の特定と修正方法に正確性と速さを向上させることで、変更時の工期を短くし、修正によるデグレーションを防止したいと思います。正確性と速さを向上できる、そんなドキュメントや、ドキュメンテーションはないでしょうか？
問題
影響範囲の把握を間違えると、修正の納期・工数に大きな影響を与える。場合によっては、デグレを含む障害発生の原因となる。
原因
影響範囲を正確・確実・迅速に把握するには、各種のドキュメントが揃っていなければならない。しかし、それらのドキュメントを目的に合致するように維持しておくことは至難であり、多くの場合は信頼に足る必要なドキュメントがほとんどない、という状況である。
解決策
・１００点満点の条件を満たすドキュメントを揃えることは諦めて、８０点のドキュメントの維持・利用を目標とする。 ・それは、ファイル（またはテーブル）とプログラムのＣＲＵＤマトリクスである。このマトリクスはファイルを縦軸、プログラムを横軸にとり、当該ファイルのＣｒｅａｔｅ，Ｕｐｄａｔｅ、Ｒｅｆｅｒ，Ｄｅｌｅｔｅを行うプログラムにその記号を入れるものである。 ・影響範囲が及ぶのは、必ず変更されたファイルを経由する。したがって、今回の修正で変更されたファイルを触るプログラムを突き止めれば、影響範囲の最大範囲が判明する。 ・しかし、ファイルを触っているプログラムが、変更になった項目を触っているのかどうかは分からない。これは、そのプログラムのソースか、詳細仕様書が信用できるのであれば詳細仕様書を解読する。 ・プログラムや詳細仕様書を迅速・正確に解読するには能力が必要であるが、影響範囲を見落とすということは起きない。 ・影響範囲の見落としが、問題を引き起こすのであるから、この方法は手間があまりかからなくて見落としを防げる、８０点の方法であると言われる所以である。 ・因みに、このＣＲＵＤ図を手作業で作成するのではなく、ソフトウェアで作成できるのが影響範囲分析ツールである。項目レベルまでの影響範囲を把握できるツールもあるが、その前提条件としてはデータ項目の標準化が必要である（データ項目名が同じでなければ、影響範囲先として引っ張り出すことができない）。

ご参考～回答例～

相談者「影響範囲の把握を間違えると、修正の納期・工数が大打撃を受けるんですよね…下手すると、デグレとか障害とか…ホント地雷です…」

回答者 A「影響範囲を正確・確実・迅速に把握するには、各種のドキュメントが揃っていないといけないですが、それらが目的に合致するように維持しておくことは至難！です！！信頼に足る必要なドキュメントがほとんどない、という状況がほとんどです！！」

回答者 B「100 点満点の条件を満たすドキュメントを揃えることは諦めて、手間があまりかからなくて見落としを防げる 80 点のドキュメントの維持・利用を紹介します。ファイル（またはテーブル）とプログラムの

CRUD マトリクスを作成するんです。このマトリクスはファイルを縦軸、プログラムを横軸にとり、当該ファ

イルの Create,Update,Refer,Delete を行うプログラムにその記号を入れるものです。影響範囲が及ぶのは、必ず変更されたファイルを経由する。そうすると、今回の修正で変更されたファイルを触るプログラムや、影響範囲の最大範囲が判明します。でも、ファイルを触っているプログラムが変更になった項目に必ず影響されるわけではないから、詳細をプログラムのソースか、詳細仕様書(※信用できる場合のみ)から調査するべきだね。プログラムや詳細仕様書を迅速・正確に調査するには能力が要るけど、影響範囲を見落としません。」

回答者 A「ちなみに、この CRUD マトリクスを手作業で作成するのではなく、ソフトウェアで作成できるのが影響範囲分析ツールです。項目レベルまでの影響範囲を把握できるツールもありますが、その前提条件としてはデータ項目の標準化が必要です（データ項目名が同じでなければ、影響範囲先として引っ張り出すことができない）。 」

結論：「CRUD を　使う哲人　クラッダー」(もしプロジェクトの女子マネージャーが…)

1.5.2. 修正方法と意図が正しく伝わらなくて…(Nさん(32 歳)、チームサブリーダー)

相談

当該システムを初めて担当するプログラマ（開発経験はそこそこある）に複数のプログラム修正を伴うプログラム変更を依頼したのですが、単純な修正（表面的なもの）にとどまっており、期待には遠く及びませんでした。

修正方法を記載したサンプルを提示し 1 本目だけは早めに行い実地指導したつもりだったのですが…きちんと成果を出させるために良い方法はないでしょうか。

※補足（具体的例）

「A を A’に変えてほしい。B、C も同様にお願いします。」といった作業指示を出したところ、B、C も全て A’に修正されてしまった。こちらとしては、B は B’に、C は C’にしてほしかった。

問題

こちらの意図した品質が確保されておらず、そのカバーリング等含めスケジュールの変更を余儀なくされ、結果的に作業が遅延した。

原因

- 作業者が本質的な変更内容を理解していなかった。また、依頼者も本質的な要件を伝えていなかった。
  （多少の期待はあったにせよ、行間を読ませる形になってしまった）
- 依頼者による成果物チェック（ソースレビュー等）が行われていなかった。
  （そのシステムに初めて触れるプログラマということで、チェックは手厚く行うべき）

解決策

まず、「作業範囲に対するレビュー」を行うことが有効な解決策となり得る。今回の例であればそのプログラマに「自分は何をするのか」を報告してもらい、その認識が正しいか否かをチェックする。これで、実施すべき作業の共通認識は確立できる。

また、作業途中でも頻繁にレビューを行い、問題の早期発見に努める。ただし依頼物件全てをチェックするのは、時間もかかるし現実的ではない。例えば成果物のうち 1 つ 2 つを抜き出してチェックし、不備があれば他のものも同様の誤りを犯していないかどうかチェックさせる、などの方法も考えられる。（犯す誤りの傾向を掴む）

レビューについては、

- 軽めのレビューを繰り返す
- 対面方式ではなく、添削方式レビューの実施
- クロスレビューの実施

などを状況に応じて使い分けることが有効になり得る。

1.6. 本番環境での作業工程

1.6.1. リリース手順がないシステムでのリプレースを実施することになりました…(Sさん(34歳)、開発担当)

相談

規模が大きく複数年にわたって機能の変更、追加を行ってきた業務システムをハードウエア、ミドルウエアの保守期限切れに伴い設備のリプレースを実施することになりました。しかし環境周りに関するドキュメントが最新化されておらず、構成管理に関しても自部門の範囲（業務ＡＰ系）についてしか実施していないような現状です。このような場合に漏れなくリプレースを行うにはどうしたらよいでしょうか？

問題

出荷後に、一部機能のリリース漏れによる障害（クレーム、問合せ）が発生する恐れがある。

原因

- 出荷手順作成時に、出荷対象の分析が出来ておらず構成管理対象からもれていた。
- 出荷手順（リリース手順）でのチェックされていない
- 他部門（システム連係先、運用部門、等）から提供されるツール類をまとめて構成管理をしていなかった。
- 運用引継ぎ書を保守ドキュメントの対象外としたためにメンテナンスがされなかった。

解決策

情報が足りない中で、『もれなく』することは非常に難しいが、以下の観点で対処する。

a.ハードウェア変更では、既存ハードウェアから新ハードウェアへの変更に伴い、変更される項目をハードウェアメーカから入手する。

b.ミドルウェア変更でも、同様にミドルウェア開発元からの変更資料を入手する。このミドルウェアがオープンソースの場合は、そのコミュニティのサイトを探し、変更に関わる影響個所を入手する。

c.上記、両資料を入手したら、当該システムでの影響を調査する。

1.6.2.リリース担当者が、退職し引き継ぎも無い状態で、リリースを担当する事になりました …(Sさん(34 歳)、開発・運用担当)

相談
突然、前担当者が退職し、何も知らない私が後任に指名された。1 ヶ月後に、当該システムの機能追加が予定されており、社内を隈なく調べたが、リリース手順に関するドキュメントが何も無い事に気付いた。
問題
障害無くリリースを行う自信が無い
原因
担当者に任せきりで、組織として管理できていなかった
解決策
ドキュメントが無いなら、人づてに聞いて回るしか手はないですね。 前担当者は居なくなったが、前々の担当者が居るなら聞いて、確認しながらドキュメント化しましょう。 前々担当者が居ない場合は、当該システムではないが、同じ環境・インフラで稼働しているシステムがあるなら、そのシステムの担当に聞きましょう。

1.6.3. ハードウェアの保守切れに伴い、ミドルウェアも入れ替える必要が出てきたが、ミドルウェアの資料が無いことに気付いた…(Sさん(34 歳)、開発担当)

相談
ミドルウェアは、オープンソースであり、旧担当者（既に退職）が、調べて導入したため、詳細資料は一切存在せず、どうやったら、無事にミドルソフトウェアのバージョンアップを行えるか教えて頂きたい。
問題
既存システムとミドルウェアの関連が見えず、障害なくリプレースを行える自信が無い
原因
担当者に任せきりで、組織として管理できていなかった
解決策
・オープンソースのサイトからバージョンアップ情報を探す ・オープンソースには、メーリングリストでのコミュニティがあるので、参加して、問合せをしてみる

1.6.4. 本番データをテストアクセスしそうになりました…(Gさん(24 歳)、SE)

相談

本番相当のデータで確認を計画しており、テストに利用するつもりでした。あやうく本番環境にアクセスするところでした。

問題

由々しき問題です。“そもそもとってもまずいじゃない”の一言です！絶対に起きないように仕組みを作っておかなければいけません。

単なるバグや障害や事故ではなく、不正アクセス/情報漏えい/守秘義務違反になりますし、告訴/訴訟沙汰まで考えられます。

原因

- コンプライアンス意識の欠如・不浸透
- 時間がないという言訳
- より本番に近いデータを使いたいという気持ち

解決策

- テスト計画時で、既に社内で通知徹底されたルールであっても、コンプライアンスに反することをリスクとして挙げ、問題ないことを確認しあう。そして、それを時間がない場合でも徹底するよう指導（声掛け）する。
- テスト計画時、テストに関する目標、テストに使うデータについても明確化し、意識合わせをする。「本番データを使うべきか」をコンプライアンスを含めて検討する。併せて、使用する場合は入手・使用後の廃棄について社内のコンプライアンス規則を遵守し管理徹底する。

ご参考～回答例～

相談者：「不正アクセスを行ってしまいそうに…あやうく会社としての責任問題となるところでした…。」

回答者 A：「由々しき問題です。“そもそもとってもまずいじゃない”の一言です！絶対に起きないように仕組みを作っておかなければいけません。

単なるバグや障害や事故ではなく、不正アクセス/情報漏えい/守秘義務違反になりますし、告訴/訴訟沙汰まで考えられます！コンプライアンスについては現場で意識を高くする必要がありますね。」

相談者：「会社で規則がないからって、意識が低かったのは反省しています。」

「コンプライアンス上問題があることはわかっていますが…本番以外のテスト環境を作る時間もお金もないんです。」

回答者 B：「時間がない場合でも、コンプライアンスは意識しなきゃダメだよ。コンプライアンスが浸透していないと、こっそり本番環境を使う人が出てくるよね。

テスト計画時に、どこまで本番環境に近い環境を用意するかとか、データについても

明確化するべきだよね。場合によっては、本番データを借用することもあるけれど、廃棄とか、後始末を社内規則によってしっかり管理すべきだよね。」
回答者 C:「個人でいろいろ判断しているように見えます。細かいことでも、チームで話して決め、責任をチームで取れるようにしましょう。」
回答者 A:「そもそもなんでアクセスできてるの？みんな管理パスワード知ってるんじゃないの？危険危険！！」
回答者 C:「対象が本番データである以上、コンプライアンスが第一です。が、この問題には運用方法の変更が伝わっていないという問題もあります。
情報共有方法を見直す必要がありますね。定例会だけでなく、メールや回覧などの複数の方法を実施してはいかがでしょうか？
運用手順書・環境構築手順書の改訂は忘れがちですが必要です。手順に従うことも必要です。
また、テスト計画の打ち合わせ・テスト計画書のレビュー時に実行する手順を確認することが重要です。
保守ですと前回と同じだから…とおろそかになりますが、手順が頻繁に代わるのも保守です。過去と手順が変わっていないかどうかしっかり確認する必要があります。」
結論： 「コンプライアンス…甘くみて　会社を滅ぼす　ラストボス」

1.7. 不具合対応

1.7.1. 不具合を恒久対応するのに必要な工数を交渉したいのですが…(Kさん(42 歳・プロジェクトリーダー))

相談
不具合発生時の対応として、次のような方針を採っています。 (1) 取り急ぎ応急措置的な対応を行い、業務を行える状態にする （緊急対応） (2) その後、改めて具体的な修正方針を検討し、対応を行う(緊急対応をいつまでも続けるわけにはいかないため、それなりに急ぐ) (2)の段階において、理想的な修正を行うだけの期間・工数を確保するにはどうしたらよいでしょうか。(確保できた工数により、再発防止策など含めた恒久対応を行うか、不具合箇所の是正に留めるか、方針が変わってくるが、できれば恒久対応が必要な場合、それを行えるだけの工数をどのように確保するか)
問題
十分な工数が確保できないと、保守者が理想と考える修正が行えない。
原因
保守者が提示した工数が認められなかった場合、顧客が期間・工数を判断するのに十分な材料が提供できていないことが原因だと思われる。
解決策
・ 緊急対応的な応急処置は、運用でカバーする範囲に留め、是正処置の検討・提案を行い、処置する。 ・ 報告資料や見積もり資料において、恒久的対応をする場合のメリット、デメリット、リスク、影響度等をお客様の業務面から説明する。 ・ 規模が小さく利益分をのせると不自然な金額になる場合は別途項目をわけてきちんと顧客に説明する方法もある。 「この項目は作業を着実に進めるために必要なコストです」 恒久対応しないにしても、影響範囲、テスト方針を顧客に納得してもらうのはもちろんとして、「今後、同じような不具合を出さないように」という対応策も一緒に提示することが必要になるであろう。(恒久対応の場合はなおさらのこと)

1.7.2. 不具合対応の工数確保に苦労しています…(Gさん(24 歳)、SE)

相談

本当は調査に時間をかけたいのですが、時間をかけすぎると、リーダー、顧客殿に不安を与えそうで…

なるべく時間を掛けたくないのですが、プログラム修正を行った場合、どの程度のテスト、成果物を準備するのが妥当でしょうか。

- 恒久対応調査とはいえ、不具合に関する対応なのであまり時間はかけられない。
- システム全体を把握できていない状況で影響範囲調査は行えるか？

問題

本件は、解決策にある、

「不具合の修正では保守者は萎縮しがちで、問題を出来るだけ小さく伝えようとしたり、時間をかけずに対応しようとしたりしてしまいがちである。」

というのが問題の本質なのだと思います。そのため現タイトルになっているのだと思います。

そのため、「保守に必要な工数の健全化」が最終的なゴールだと思います

原因

「顧客に不安を与えない」ためには、システム全体を把握せずに影響範囲調査を行ってはいけない。

しかしながら全体を把握するのが非現実的であることは十分に考えられ、その矛盾を解決できないため相談が起きていると思われる。

とかく不具合の修正では保守者は萎縮しがちで、問題を出来るだけ小さく伝えようとしたり、時間をかけずに対応しようとしたりしてしまいがちである。

しかしこのような対応は、不具合を表面的な修正で終わらせてしまいがちである。

問題が発覚したからこそ、生じうる不具合や原因を正確に捉え、ミスが混入しないように慎重に作業する必要がある。

解決策

不安を与えないためには、工数が必要であることをお詫びと共に顧客殿にきちんと説明すべき。

対応を急ぐのであれば、緊急対応として対策すべき。

顧客のシステムを止められないなど、緊急の対処の必要がある場合には、

暫定対応と恒久対応にわけ、最終的には根本的に対策する。

バージョンアップする時間が無いなどの理由で恒久対応を拒否されることもあるが、説明し準備しておけば、次回再発したときにすぐに対応できる。

また、調査効率の向上には、影響範囲調査が効率的に出来るよう、システム概要図（ER 図）のようなシステム全体が分かる資料があるとよい。

影響範囲調査ツールなども有用である。

1.8. 引継ぎ

1.8.1.リーダー交代の引き継ぎが難航します…期間や時間に問題がある場合、どうすればよいのでしょうか？(K さん(42 歳・プロジェクトリーダー))

相談

では本日の相談は、神田生まれのプロジェクトマネージャ、K さんです。

Kさんは何度かプロジェクトの引き継ぎを体験していますが、いつもうまく行かず、悩んでいます。

引き継ぎの問題は多岐にわたりますので、今日は引き継ぎ時間の確保の問題と、それが不十分だった時の対処の問題について、伺ってみましょう。

では、K さん。ご相談をどうぞ。

「リーダー交代って、いつもそうですけど、時間が確保できないんです。

マネージャーですから、自分で時間は管理するんですけど、いつもプロジェクトの作業で一日終わってしまうので、引き継ぎを行うにしても、受けるにしても時間が足りません。部下なら時間を作ってあげられるんですけどね。

まあ、いつもは多少は時間は取れるんで、資料のありかとかは分かるんですけど…

今回は特にひどくて、前任者がいきなり退職とかで…。有給取得に入っちゃってまともに引き継げないまま着任してしまいました。」

問題

引継ぎの期間や時間に関する問題としては、以下が考えられる。

・急な引き継ぎで期間が不十分である。

・前任者・後任者それぞれについて、業務と並行して行われ時間が確保できない。

・前任者の休暇など、必要十分な作業時間が確保できない。

原因

急な引継ぎはもちろんのこと、予定されていた引継ぎであっても時間が不足して十分に引き継ぎができないことがある。

背景には、業務内容が十分に整理されていなかったり、されていても個人に頼った業務になっていて、引き継ぐことが想定された手順になっていないことがあげられる。

保守開発においては、汎用的な業務手順を作るよりも個人に頼ったほうが目先の対応は早いことがしばしばあり、引継ぎの頻度が少なければ個人に頼るほうが合理的でさえある。引き継ぎの問題はこの手段をとった場合に現れるリスクである。

解決策

予防

”完全な”業務手順を作ることを考えがちだが、保守の場合は合理的でない事が多い。

引継ぎ時間を十分にとって、実際に業務をしながら引き継ぐ方が引き継ぎ漏れが少なく、確実な方法である。

また、予め個人依存を弱めるよう、（マネージメントの仕事であっても）複数の人間で対応できるようにしておくと、急な異動にも強い組織を作ることができる。

対症療法

引き継ぎの準備が不十分な状態でこの事態が発生した場合、十分な引き継ぎは不可能と考えて、以下の対策を行う。

・最小限の引継ぎ

確保できる時間内で可能な最小限の引継ぎを行う。

この引継ぎは、必要な業務をリストアップして行う。組織にも依るが、概ね過去１年程度や？の帳票やメールを見れば業務はひと通り網羅できるだろう。

その中から、重要な作業アイテムの順に引き継ぎを行う。

その他のアイテムは、１年分の帳票やメールがあれば、過去にどのようにしたかのヒントにはなる。

・バックアップ体制を検討する。

前任者や業務を理解している人が組織内にいるのであれば、その後もフォローできるよう連絡手段を確保しておく。

そういった人がいなければ、「どうしたらいいかわからない事態」に対して過去にとらわれず、その時点でベストの対処を検討できる仕組みを検討しておく。

ご参考～回答例～

回答者 A：「時間が足りないかあ。よくありますね。まず、どれぐらい時間があれば足りるんでしょうか？」

相談者：「そうですねえ。プロジェクトによって規模が違えば、前任者の資料の残し方も違うので、一概にはいえないです。今回の例で言えば、前任者が『資料は全部あるから、読めばわかる。』って休みに入っちゃって。あとから考えると随分しっかりした資料ではあるんですけど、事実関係の確認とかで時間取られちゃいました。資料がしっかりしていたとしても、ひと通りの申し送りは必要ですね。」

回答者 B:「資料の申し送りであれば、資料の量によりますね。キャビネット見れば見積もれるかな？でもあんまり多すぎると全部は見ないなあ。そうすると前任者は情報を整理する工数がかかるわけか。」

回答者 A:「資料が揃っている例なんて、少ないんじゃないですか？」

相談者：「そうですね。以前だと、なーんにも資料残さないで引き継ぎされたこともあります。その人は社内の異動だったのでなんとかなりましたけど。その場合は何か事が起こってから聞くことになるので、メンバやお客様に迷惑をかけっぱなしでした。まあ、そのおかげでリスクマネジメントの重要性を学びましたけどね（笑」

回答者 C：「いますよね。『なんかあったら聞いてくれ。じゃ！』っていう人（笑」

回答者 B:「そういう人って、そのほうが楽だと思ってますよね。ほんとに楽なのかなあ。」

相談者：「時間だけ考えれば必要最小限で済むって思ってたこともありますけど、リスクが大きすぎますよ。毎日ハラハラします。精神衛生上良くない。」

回答者（発想広げる役）「でも、資料あればいいってもんでもないじゃん。ゴミみたいな資料渡されても困るよ。」

相談者「ああ、そういえば先に『資料残さなかった人』と言いましたが、彼なりの資料はあったんです。でもメモみたいなもので理解不能。断片的でバラバラ、使えなかったんで、実質『ない』状態でした。彼にしてみれば『資料はある』だったのかもしれないですけど、質問しても『〜に書いてある』とは言って来なかった。本人もわかると思ってないですよ。」

回答者（進行役）「えーと、資料については別に議論するので、時間の問題に戻しましょう。

今のところ、資料がある場合は引き継ぐ資料の量に比例するって話がでてます。

あと、資料が不十分な場合には、どれぐらい時間が必要でしょうか。」

全員「…」

回答者（経験のある人）「…実際、資料がそろっていてもそろってなくても、引継ぎはある程度のところで割り切って線引きする必要があります。

資料が十分でも、引き継げていないことは案件実行時のリスクとして組み入れることになる。

資料が不十分であればそのリスクが大きくなるのだと思います。

これは資料だけではないですね。引継ぐ人は経験が不足だったり、人的つながりが薄かったりします。こういったことは結局リスクに還元することになります。」

相談者「減らしたいのはそのリスクです。」

回答者（進行役）「えーと、今回はあくまで引き継ぎ時間の確保についての話で…（汗」

回答者（経験のある人）「引き継ぎ時間を確保するにはどうしたらいいか。という話と、引き継ぎ時間が少ないときに、優先的に何をすべきか。ですね。（笑）

前者は単純に計画力の問題です。確保しろ。としか言いようがない。

後者は時間がない中でリスクを最小限にする。ってことになります。その話です。」

回答者（経験のない人）「そのリスクを減らすためには、あらかじめ業務手順やスキルを明確にしたりする必要がありますね。人のつながりは…どうしよう。」

回答者（進行役）「自分の業務手順やスキルってなかなか明確にはできないよ。『おまえの脳内を書け！』っていわれたことあるけど、そんなの無理（笑）」

回答者（経験のある人）「個人の手順を明らかになんて無理です。

時間がないなら、優先すべきは作業の全体像でしょう。

組織の手順を書いたほうがいい。最優先は組織が実行するタスクの種類。そしてその手順。これは作業の重要性が高いものが優先になります。それにしたって組織が実行するタスクの種類がないと優先順位のつけようもない。

万一これがないのなら、引継ぎはまずはこれを話題にすべきです。

これは日常業務においても明確にしない弊害のほうが多いので、引継ぎが生じていない組織もすぐ書き始めたほうがいい。

これで作業の全体像がわかれば、個人はそれに合わせて作業できます。引き継いだ人も、全体の作業の流れがわかれば聞くべきところもわかる。」

相談者「組織が実行するタスクの種類って、短い引継ぎのなかで洗い出せますかね？」

回答者（経験のある人）「あんまり細かいことかんがえても無理です。まずは単純に顧客からの依頼書、契約書など、顧客とのやり取りを並べてはいかがでしょうか？いくらなんでもこれはあるでしょ？そのなかから重要なものから順に取り上げていけば、手順って思い出せますよ。で、『説明できなかったもの』も明らかになる。これが大事です。」

1.8.2.リーダー交代の引き継ぎが難航します…これって前任者と後任者のスキルの差なのかなぁ(K さん(42 歳・プロジェクトリーダー))

相談

リーダー交代の引継ぎがスムーズにいきません。

引継ぎ情報がまとまってないというよりは、プロジェクトマネージャの前任者と後任者のスキルの差異があるように思います。

- 後任者の経験が少ないと実践ノウハウが無いので引き継ぎの要点が理解出来ない。
- 後任者にないスキルに頼っていた業務があり、後任者が実行できない。

問題

プロジェクトマネージャの前任者と後任者のスキルの差異については、以下の問題が考えられる。

- 後任者が新任のプロジェクトマネージャであり、実践ノウハウが無いので引き継ぎの要点を理解出来ない。
- 後任者にないスキルに頼っていた業務があり、後任者が実行できない。

原因

スキルは人それぞれ違うが、引継ぎにおいてはそれが考慮されないことが多い。

また、やったことがない業務であれば、スキルが有るかどうかすらわからない。

解決策

予防

前任者と後任者がしっかりコミュニケーションして各自のスキルを確認する。このためには、具体的な業務手順を詳細に確認する必要がある。

後任者が未経験の場合には、出来るだけ長い期間を確保してOJTのなかで引き継ぐことが望ましい。

スキルに大きな差異がない場合は問題ないが、差異がある場合には、他の人に助力を求めたり、後任者が実行する場合の業務手順自体を再構築することになる。

これはある程度期間の必要な作業となるため、引き継ぎ期間内にすませる内容と、引き継ぎ期間後に実施する内容を区別し、前任者またはこれに準ずるメンバによるフォロー体制を築く。

対症療法

引き継ぎを終えて、スキルの不足により業務が思うように進まない場合には、プロジェクトとしてどのように対処するか検討する。

早い段階でその仕組みを構築しておく。

その業務の目的を理解することで、どう対処すればよいか検討できる。そのためにも、業務手順はその目的をしっかり確認しておく。

ご参考～回答例～

相談者：「スキルは人それぞれ違うが、引継ぎにおいてはそれが考慮されないことが多いんですよね。未経験業務だと、スキルが有るかどうかすらわからないし…でも普通、若手に引き継いだらスキルはないのが普通か…。(愚痴モード)」

回答者 A：「やっぱコミュニケーションですよ！前任者と後任者がしっかり話をして各自のスキルを確認しなきゃ。そのためには、具体的な業務手順が詳細化されている必要がありますね。」

回答者 B：「後任者が未経験の場合には、出来るだけ長い期間を確保して OJT のなかで引き継ぐことが望ましいと思います。」

回答者 C：「スキルに大きな差異がある場合には、他の人に助力を求めたり、後任者が実行する場合の業務手順自体を再構築することになるんじゃないでしょうか。期間が必要となるため、引き継ぎ期間内、期間後に分けて実施内容を区別し、フォロー体制を整えるのも重要ですね。」

回答者 D：「引き継ぎを終えた後も、業務がスムーズに進まない場合には、プロジェクトとして対処する方法を検討しておくといいですね。早い段階でその仕組みを構築しておくといいです。」

相談者：「具体的には…？」

回答者 D：「その業務の目的を理解することで、どう対処すればよいか検討できることがあります。そのためにも、業務手順はその目的をしっかり確認しておくことが必要です。」

1.8.3.リーダー交代の引き継ぎが難航します…これって業務の範囲、作業量が問題なのだろうか(K さん(42 歳・プロジェクトリーダー))

相談

リーダー交代の引継ぎがスムーズにいきません。

引継ぎ情報がまとまってないというよりは、引き継がれる業務の範囲、作業量についての問題があるのではないでしょうか。

- プロジェクトの変化に伴う引き継ぎの場合、業務範囲も変化するが、引き継ぎ後ににしか実態が分らない。
- 後任者の既存の業務は継続され、作業追加となって工数がひっ迫する。

問題

引き継がれる業務の範囲、作業量についての問題は、以下が考えられる。

- プロジェクトの変化に伴う引き継ぎの場合、業務範囲も変化するが、その対応は引き継ぎ後に”臨機応変に”対応するとされ明確になるまで業務手順が安定しない。
- 後任者の既存の業務は継続され、作業追加となって工数がひっ迫する。

原因

マネージメントなど、組織に大きな影響を与える人の入れ替えは、役割分担や、やり取りの変化も発生するが、引継ぎにおいてそこまで考慮されていない。

解決策

予防

保守では、プロジェクト全体の引き継ぎでは業務範囲が変化することが多く、個人の引き継ぎでは作業の追加となることが多い。

これらは、引き継ぎ計画の時点で、当事者以外のメンバーを含めた役割分担や業務手順の再構築を計画する。

対症療法

予期しない業務範囲の変化や予定の超過は常に発生しうるので、引継ぎ後に業務が安定するまでは、進捗をよく確認してその可能性が生じれば対策を講ずる。

ご参考～回答例～

相談者：「マネージメント担当者の入れ替えは、プロジェクトに大きく影響するんですよね…やり取りも変化するのに…引継ぎではあまり考慮されないんですよね…。(愚痴モード)」

回答者Ａ：「保守では、プロジェクト全体の引き継ぎでは業務範囲が変化が、個人の引き継ぎでは作業の追加が多いんですよね…引き継ぎ計画の時点で、当事者以外のメンバーを含めた役割分担や業務手順の再構築を計画してはどうでしょう。」

回答者Ｂ：「予期しない業務範囲の変化や予定の超過は常に発生しますよ！引継ぎ後に業務が安定するまでは、進捗をよく確認して対策を講ずるしかないですね。」

1.8.4.リーダー交代の引き継ぎが難航します…これって引継ぐべき情報が問題なのだろうか

(K さん(42 歳・プロジェクトリーダー))

相談

リーダー交代の引継ぎがスムーズにいきません。

引継ぎ情報がまとまってないというよりは…やっぱり引継ぎに必要な情報を用意してあるつもりであっても、すべてを完全に用意することはできないのかもしれません。

暗黙知として隠れてしまっている情報もあるのかもしれません。情報不足の場合にどう対処すればよいのでしょか。

問題

引継ぎに必要な情報を用意してあるつもりであっても、すべてを完全に用意することはできない。

情報の中には暗黙知として隠れていた情報もありうる。

問題は、情報不足の場合に対処できるかどうかである。

原因

業務手順や必要な情報が明らかになっていない。

スキルや組織の変化などにより、必要な情報が変化した。

解決策

予防

可能な限り作業を明確にしておいた上で、引き継ぎにおけるヒアリングを行う。

組織形態にもよるが、進捗管理の中で必要な情報はある程度拾えるはずである。

過度の個人依存をしている組織の場合、作業内容はなかなか明確にならない。この場合は OJT 方式による引き継ぎを主とする必要があるため、期間を十分に確保する。

隠されていた課題の抽出は困難である。ヒアリングなど個人間の信頼関係によって抽出する。

対症療法

進捗をよく確認して必要な情報があればできるだけ速やかに情報を収集する。

.

ご参考～回答例～

相談者：「業務手順や必要な情報が明らかになっていない…かもやっぱり…。スキルや組織の変化などにより、必要な情報だって変化しちゃったんだ…orz。(愚痴モード)」

回答者 A：「可能な限り作業を明確にしておいて、引き継ぎにおけるヒアリングを行ってみてはいかがですか。組織形態にもよりますが、進捗管理の中で必要な情報はある程度拾えるのではないでしょうか。」

相談者：「具体的にどんな感じですか？」

回答者 B：「属人的傾向が強い組織の場合、作業内容はなかなか明確にならないんですよね…そうなると十分に期間をとって OJT！これしかないですよ！課題が隠れている場合は客観的に抽出するのは難しいので、ヒアリングするしかないですね。」

第 21年度(2011年)SERC B グループ作業部会報告書

保守作業改善の基盤技術調査

B グループメンバー （順不動）

(株) NTT データ	峯村　圭介
(財)経済調査会	押野　智樹
東芝ソリューション(株)	川上　康史
(株)日立ソリューションズ	木部　俊之
(株)中電シーティーアイ	江尻　武志
(株)中電シーティーアイ	諸岡　隆司
(株) SRA	石川　雅彦
(株) SRA	岸田　孝一
(株) SRA	方　　学芬
(有)ウィルビーネットワーク	松尾　好博
	小林　　允

目次

Ⅰ はじめに 117
Ⅱ 研究報告 118
２．１　研究１：ＣｏｄｅＣｉｔｙ 118
はじめに 118
１．CodeCity 概要 118
１．１　はじめに 118
１．２　概要 118
１．３　都市メタファ 118
２．CodeCity による設計品質評価 122
２．１　はじめに 122
２．２　背景 122
２．３　設計調和 122
２．４　不調和マップ(Disharmoniy Maps) 124
２．５　クラスレベル設計不調和 126
２．６　メソッドレベル設計不調和 131
３．CodeCity の効果：ソフトウェア進化をたどる 133
３．１　はじめに 133
３．２　背景 133
３．３　事例研究 133
３．４　時間を振り返る 134
３．５　疎粒度表現 135
３．６　細粒度表現 139
３．７　結論 146
４．　References 146
２．２　研究２：ソフトウェア可視化の歴史 147
Ⅲ　活動報告 165
（1）　例会 165
Ⅳ　今後の予定 165

Ⅰ はじめに

ソフトウェア資産の増大とソフトウェアライフサイクルの長期化、またソフトウェア実行環境における外部モジュールへの依存度の増大により、ソフトウェア保守エンジニアの対応範囲は増大化複雑化の一途をたどっています。

それに加えて、一旦リリースされたソフトウェアは容易に廃棄されることは少なく、利用される方々の継続的な要望に応えるために絶えざる進化を要求されています.

このような状況に対処する方策として、わたしたちのグループは、保守作業改善の基盤技術に注目しました. 保守作業改善の基盤技術は、ソフトウェア進化・保守を支援する、モデル、ツール、ベストプラクティスが含まれます.

保守作業改善の基盤技術は、支援環境と密接に結びついています. 支援環境とは、豊富な計算機パワーや、インターネットと接続された環境を含みます. これらの基盤技術と支援環境を背景にしてわたしたちのグループでは、保守・開発に携わるソフトウェアエンジニアの能力を何倍にも強化(enhance), 増幅(enpower)し、自律したソフトウェアエンジニアの実践を通じてソフトウェアの進化・保守を促進するための研究を行うことを方針としています.

Ⅱ 研究報告

2．1 研究1：CodeCity

はじめに

本章では ソフトウェアツール「CodeCity」の研究成果を示す. CodeCity は Lugano 大学の Richard Wettel と Michele Lanza によって作成された, ソフトウェアシステムを視覚的に表現するためのツールである. まず最初に、CodeCity の概要について示し, 次に CodeCity による設計品質評価方法と効果を示す. 最後に CodeCity を使ったソフトウェア進化分析 について説明する.

１．CodeCity 概要

1．1 はじめに

この章では、論文[1][2][3][4][5][6][7][8][9]の内容に基づき、CodeCity の概要、特にソフトウェア視覚化と対話性の概要について説明する.

1．2 概要

ソフトウェアの視覚化は,ソフトウェア理解促進やソフトウェア進化分析において利用される一般的な技法である. CodeCity は，ソフトウェア分析タスクを支援するための三次元視覚化ツールである。 CodeCity は大規模なオブジェクト指向ソフトウェアシステムに適用可能であり，都市メタファを用いることにより，ソフトウェアシステムを三次元的に視覚化する. つまり，分析対象ソフトウェアシステムを三次元都市に変換する.

1．3 都市メタファ

都市メタファの意義

論文[3] によれば，都市メタファを採用する意義は；

・都心領域と郊外を持つ都市は，なじみのある概念である

・都市，特に大きな都市は，本来複雑な構築物なので段階的に調査して初めて理解できるものである. 同様に，われわれが複雑なシステムを理解する場合も，理解は段階的に進むものである. すべてを単純化し過ぎるメタファ（例えば，全システムまたは各パッケージを大きな立方体や球面で表すこと）はソフトウェアシステムの複雑さをうまく表

すことができず，不正確な単純化し過ぎに導く．　われわれは，ソフトウェアは複雑であるという事実をうまく処理しなければならない．

・　クラスと，クラスが配置されるパッケージの区画は，オブジェクト指向パラダイムにとって重要な要素である．そのため，これらの関係の理解は、システム全体の理解の発端となる．ただし，プログラム理解の最初の時点では，詳細の理解は不要であるため，クラス内部は表示しない．

三次元の都市に変換されたソフトウェアシステムの例として，ArgoUML の都市を図 1 に示す．

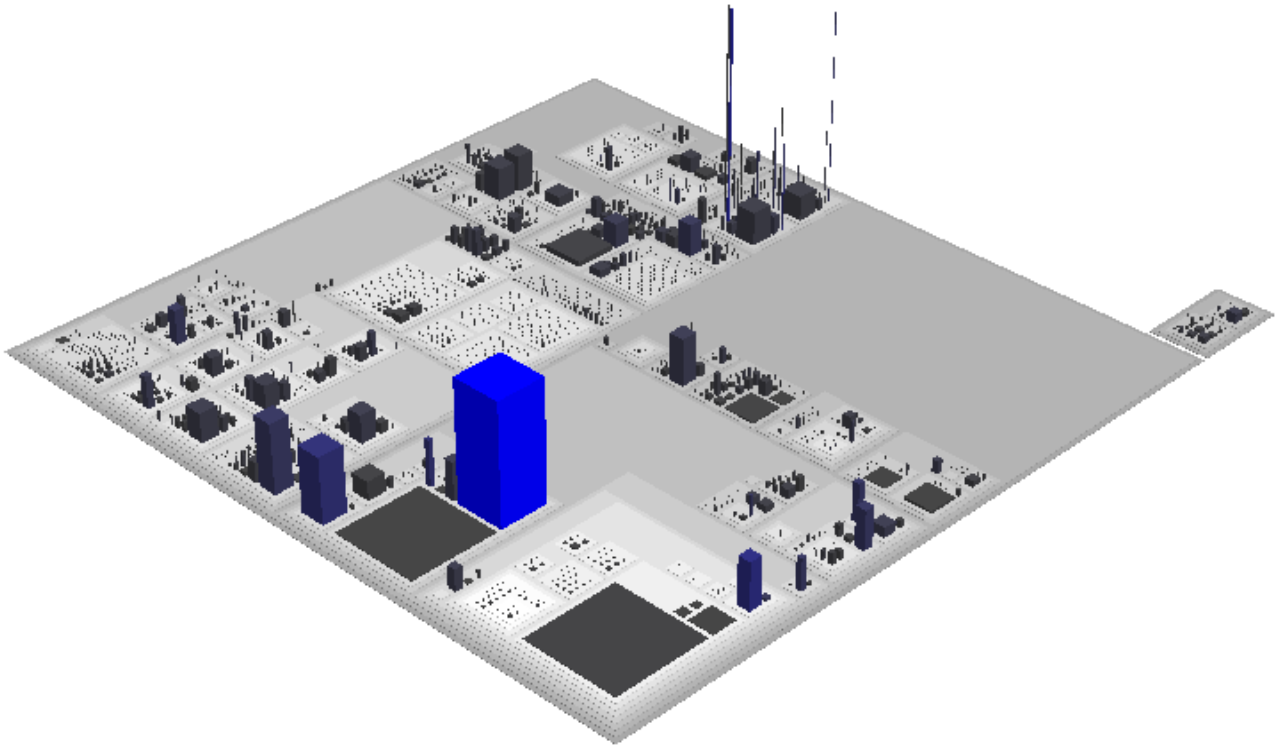


図 1　ArgoUML 0.28 CodeCity

これは，ArgoUML 0.28 の CodeCity である．ArgoUML 0.28 は，160kLOC 以上のコードを持つ Java システムである．

都市メタファの中で，パッケージは区画に変換され，クラスは区画の中のビルディングとして描かれる．クラス内のメソッドの数(NOM)は建物の高さに，属性(attribute)の数(NOA)はビルの床面積に，コード行数(LOC)は建物の色として表わされる．ビルの色が暗灰色に近いほど LOC が少なく，青い色のビルはＬＯＣが多いことを示す．

視覚化によって，あるパターンに容易にスポットを当てられる．例えば，二つの大きなビルがある（これは潜在的なゴッドクラスである）またはあるアンテナのような形の建

造物がある駐車場(parking lots)のように見える多数のクラスがある，そして小さな家がたくさんある．CodeCity の都市は，キーボードを使って対話的に操作できる． 例えば，都市の詳細に容易にズームインでき，またはある特定の区画に焦点を当てることができる．

図 1 に示した ArgoUML の都市を更に詳細に見ていく．

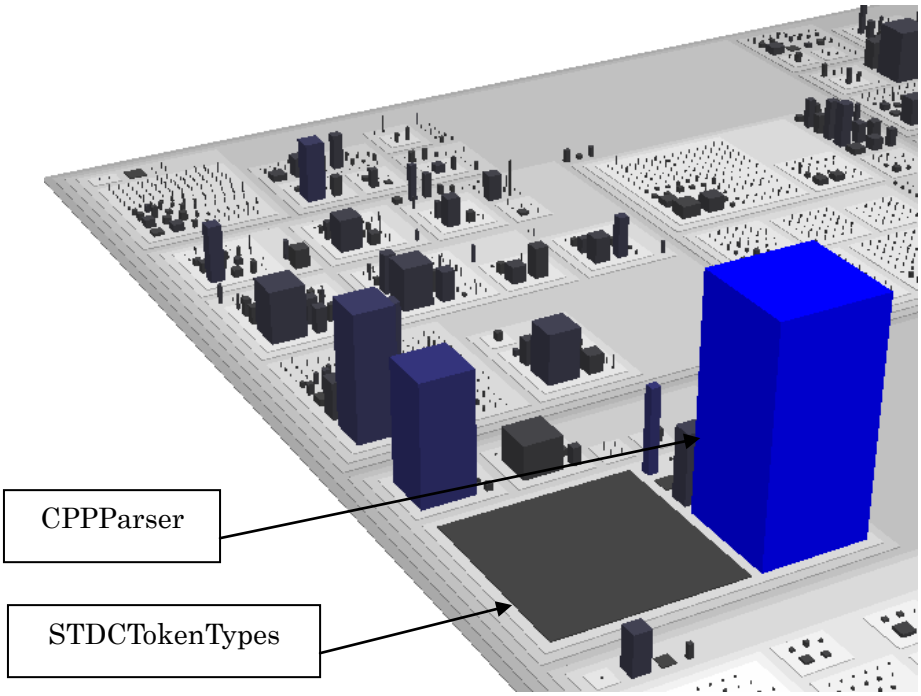


図 1 -2：org::argouml::language::cpp::reveng

パッケージ org::argouml::language::cpp::reveng 上の青い大きなビルは CPPParser である．このビルにマウスカーソルを当てると，ビルの情報が CodeCity のウインドウ右上に表示される（表 1）．また，青いビルの左側にグレーの駐車場のように見える建物がある．この建物にマウスカーソルを当てると，表 2 の情報が表示される．

class CPPParser in org::arguml::language::cpp::reveng color: 9108 lines of code height: 200 methods length: 85 attributes width: 85 attributes

表 1 ：CPPPaeser

class STDCTokenTypes in org::argouml::language::cpp::reveng color: 0 lines of code height: 0 methods length: 152 attributes width: 152 attributes

表 2 ：STDCTokenTypes

表示された CodeCity は　キーボードの操作によって，拡大、縮小、回転ができる。
次の図は、図 1 の ArgoUML の CodeCity を 180 度回転させたものである．

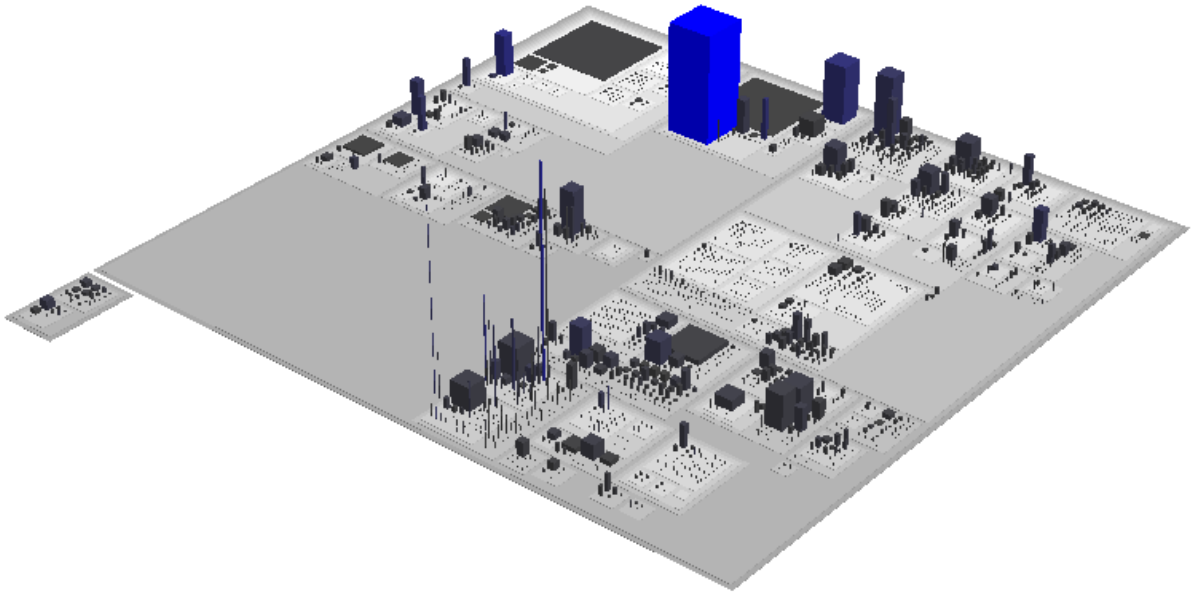


図 1 -3：ArgoUML　0.28 CodeCity(180°回転)

また，以下の図は，ArgoUML を真上から見たものである．このようにして，都市に変換されたシステムを様々な角度から眺めることにより，システムの概要を掴んだり，一部分を拡大して，特定のパッケージ、クラスの詳細を調べることができる．

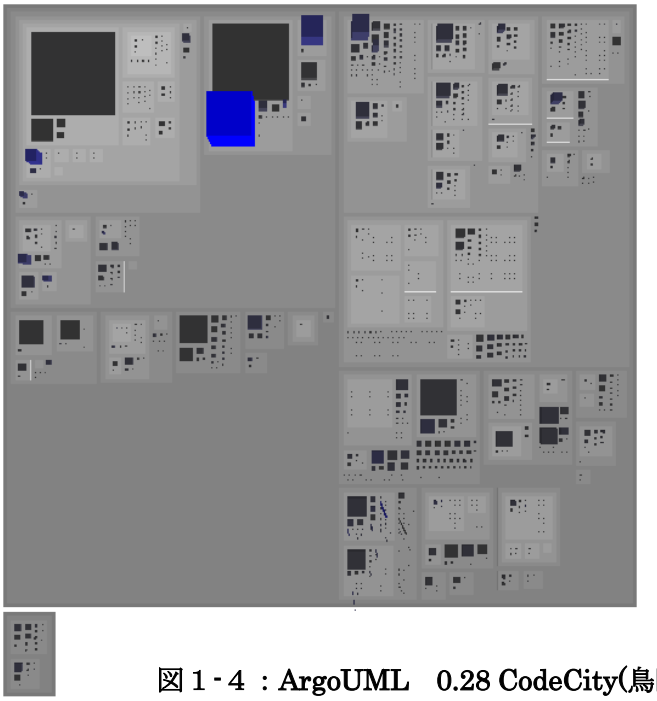


図 1 - 4 ：ArgoUML　0.28 CodeCity(鳥瞰図)

２．CodeCity による設計品質評価

2.１ はじめに

ここでは主に論文[5] の内容に基づいて, CodeCity による設計品質評価の詳細について記述する.（掲載した図表も, 論文[5]より引用）

2.２ 背景

複雑なソフトウェアシステムの設計は難しいタスクであり, その難しさを克服するために, 過去二十年以上に渡って, 多くの設計ガイドラインが作成され提出されてきた. 例えば Gamma らが 1995 年に提出したパターンの概念や, Riel が 1996 年に提出したヒューリスティクスの概念などである. これらの努力にも関わらず, 外部要因の変化つまり新しい要求の発生に起因する環境の変更によって, 当初は最良だった設計さえも, 長い時間の間に品質が低下していく. これらの品質低下プロセスに注目すれば, ソフトウェア保守と進化のために, ソフトウェアコストの 90%が費やされていると言われても驚くには当たらない. リエンジニアリングの目標は, システム設計の改良であり、将来の変更に対して進んで対応可能にさせる. しかしながら、問題を引き起こしそうなアーティファクト全てを対象にするようなものではない. そうではなく, それらアーティファクトの中で優先度の高いものに狙いを定めて対処できなければならない. その観点から, どの部分をリエンジニアリングすべきかという意志決定に先立ってソフトウェアの設計品質を評価する手段が必要である.

2.３ 設計調和

設計品質は，ソフトウェアシステム分析の関心のひとつである．この設計品質は，ソフトウェア理解(comprehensibility)と，ソフトウェアの生存期間を通じて要求される保守の総量の両方に影響する．設計を評価するアプローチのひとつは,設計調和(Design harmony)とその対語である設計不調和(Design disharmony)の概念を中心としている.

２．３．１　設計不調和の検出

1. 同定調和(Identity harmony)

同定調和は,「自らをどのように定義するか？」という問いに変換される.
ソフトウェアシステムの各要素(entity)は，その存在を正当化しなければならない.
その要素はは特定の概念を実装しているか？どのように実装しているか？実装に過不足はないか？調和の反対語は不調和であり、本論文のコンテキストでは，以下の同定不調和に焦点をあてる.

・ゴッドクラス(God Class) ：

ゴッドクラス は，自分であまりに多くを実行するクラスである．.
他のクラスとあまり協調しないが他のクラスのデータを使用するクラスである.

・ブレインクラス(Brain Class)：
　ブレインクラスは，知性の過剰な量を集めすぎるクラスであり，通常，いくつかの Brain メソッドの形をとる

・データクラス(Data Class)：
　データクラスは，複雑な機能性がない，沈黙のデータ保持クラスで他のクラスがこのクラスに依存する．
・ブレインメソッド(Brain Method)：
　ブレインメソッドは，クラスの機能を集中させる傾向のあるメソッド
・属性・操作の横恋慕(Feature Envy)　：
　属性・操作の横恋慕は，自分のデータよりも他のクラスのデータに関心があるメソッドである．

2. 協調調和(Collaboration harmony)

協調調和は "どのように他と協調するか？"という問いに翻訳できる．
各エンティティはタスクを完遂するために他と協調する．
・結合度の集中(Intensive Coupling)：
　システムの中のほんの２,３のクラスの中で多くの操作と結びついているメソッド
・ 結合度の分散(Dispersed Coupling)：
　結合度の集中(Intensive Coupling) と補完的.システムを通じて多くのクラスに分散した多くの操作と結びついているメソッド
・ 変更の分散(Shotgun Surgery※):
　あるメソッドでのある変更が他のメソッドとクラスの多くの変更を暗示するという事実を参照している．(※散弾銃で受けた傷を手術するイメージから来ている)

3. 分類調和(Classification harmony)

祖先と子孫との関係においてどのように自分自身を定義するか？
このハーモニーは,継承の文脈において二つの他のハーモニーを組み合わせる．
※この調和についてはこれ以降言及しない．

例：ゴッドクラス不調和

この設計欠陥は,最初に Riel によって指摘されたものであり,あまりにも多くの
知識が合体する傾向にあるクラスを参照する． 特徴は以下のルールによって記述される．

(1) 直接アクセスするかまたはアクセサメソッドによって,単純なクラスに過度にアクセスする

(2) 大きくて複雑である

(3) 多くの非コミュニケーティブな振る舞いをする.例えば,そのクラスに属するメソッド間で，低凝集度(low cohesion)である.

2．4 不調和マップ(Disharmoniy Maps)

対象とするソフトウェアシステムに発見戦略を適用した結果，設計不調和が得られる．この設計不調和は，深刻なものから軽度のものまで様々なレベルのものがある．レベルに応じて彩色し，CodeCity 上のアーティファクトに反映させることにより、不調和マップが得られる．換言すれば、不調和マップとは，大きなシステムにおいて欠陥のあるソフトウェア生成物(クラスやメソッドなど）を都市メタファ上に現出させるための，視覚化技法である． 設計調和を乱すアーティファクトに対しては、明るい色を割り当てて、その存在を際立たせるようにする．逆に，影響がないか、ほとんど影響がないアーティファクトに対しては灰色でシェーディングすることにより目立たなくする。
このような操作の結果として得られる視覚化情報を不調和マップと呼び，不調和マップは設計欠陥に焦点をあてるものとなる.

不調和マップの主な利点は,システム設計の概要を提供すること,そして観測者に対して,不調和が影響する実体(entity)を CodeCity 内の位置(location)に対応づけることである.

図は，ゴッドクラスに焦点を当てた JDK の CodeCity である.

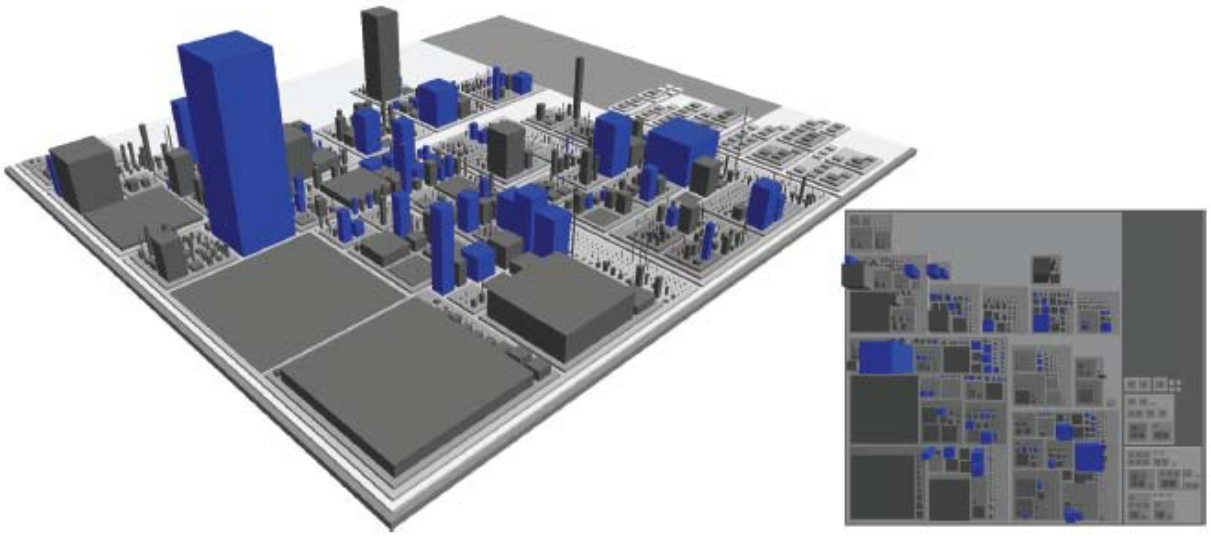


図 2-1：City of JDK with focus on God Classes

以下の図は，テキストベースのアプローチを使ってこのゴッドクラスを表現した例である．右に全クラスリストがリストアップされ，左側にはゴッドクラスが反転表示されている．

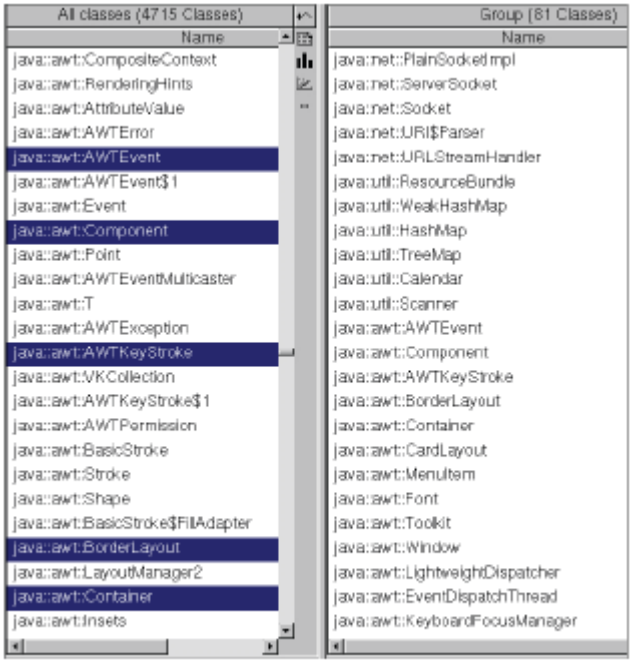


図 2-2： God Classes in JDK core (text base)

2．5　クラスレベル設計不調和

図２－３は JDK1.5 のクラスレベルの設計不調和を視覚化したものである．

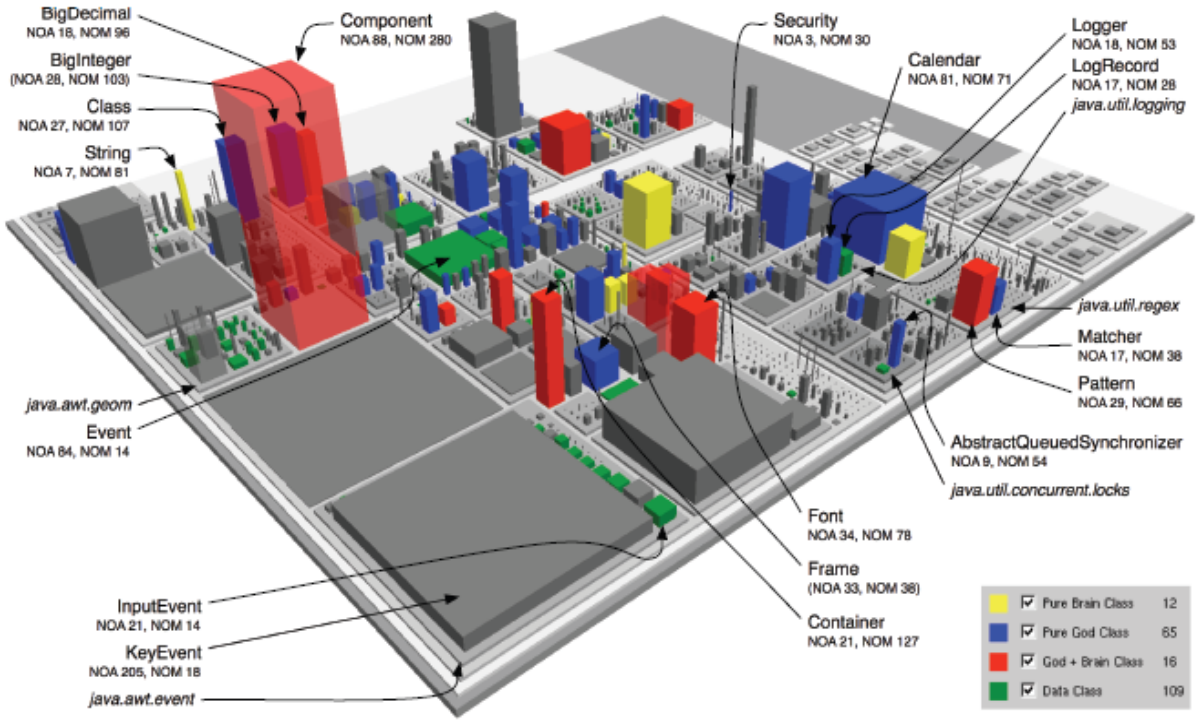


図 2-3：　クラスレベル設計不調和(JDK)

詳細に入る前に，概観を見ておくと、このシステムには多くの不調和があるにも関わらず、全体としてはよく組織化されているように見える．緑の区画にはデータクラスがあり，ゴッドクラスとブレインクラスが定義されているいくつかの場所は、複雑なクラスの区画である．

java.awt.event パッケージを表わす区画が興味深い．

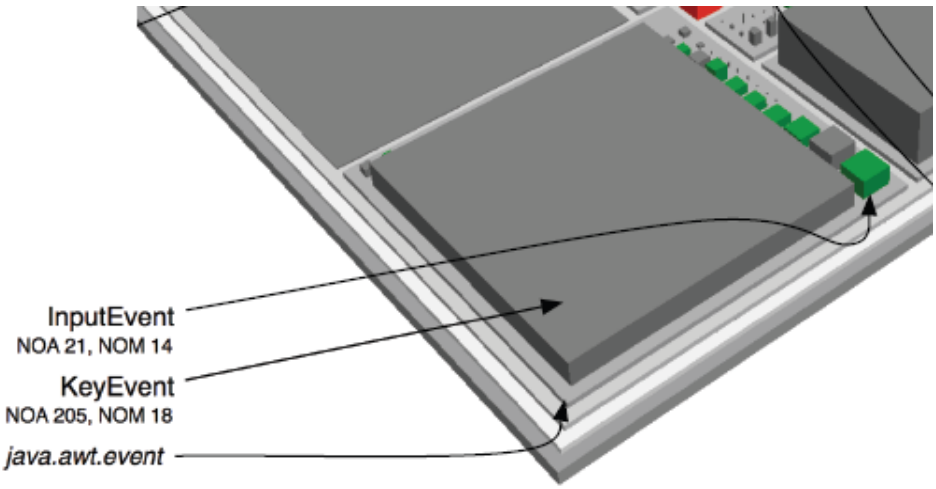


図 2-4： クラスレベル設計不調和(java.awt.event)

この区画には，大きく平坦なビルが乗っているように見える．

このビルは KeyEvent クラスと多くの小規模なクラス，InputEvent などの KeyEvent 以外のイベントクラスを表わす．KeyEvent クラスには 205 の属性があるが，メソッドは 18 しかない．そのためこのクラスをデータクラスに分類したくなる．しかし実際には KeyEvent クラスは、設計不調和とは関係のない数少ないクラスの一つである．なぜなら，このクラスの 18 メソッドはアクセッサメソッドではないばかりでなく、その中のいくつかが非常に複雑だからである．

大きな赤いビルの左側に java.awt.geom パッケージがあり、その上に緑のビルがあるのがわかる．

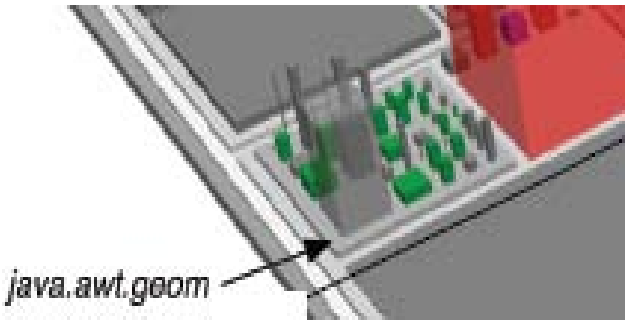


図 2-5： クラスレベル設計不調和(java.awt.geom)

データクラスは，JDK 全体では 109 あるが，そのうちのかなりの数がこの区画に集中しているように見える．この区画のスーパークラスは Event クラスであり，Event クラスは 84 アトリビュート，14 メソッドを持つ大きなデータクラスである．Event クラスは java.awt パッケージ(親パッケージ)に定義されている．ゴッドクラスでありブレインクラスでもあるクラスは，その多くが java.awt パッケージで定義されている．java.awt パッケージは Java におけるグラフィック部のコアとなる部分である．

２．５．１　ArgoUMLのクラスレベル設計不調和

AlgoUML には 17 のブレインクラス(黄色)と 33 のゴッドクラス(青)があり， 33 のうち 9 は ブレインクラス，ゴッドクラス両方の設計不調和クラス(赤)である．

17 のデータクラス(緑)は全体に分散しているわけではないが,図２－６に示されるように,まばらにある．

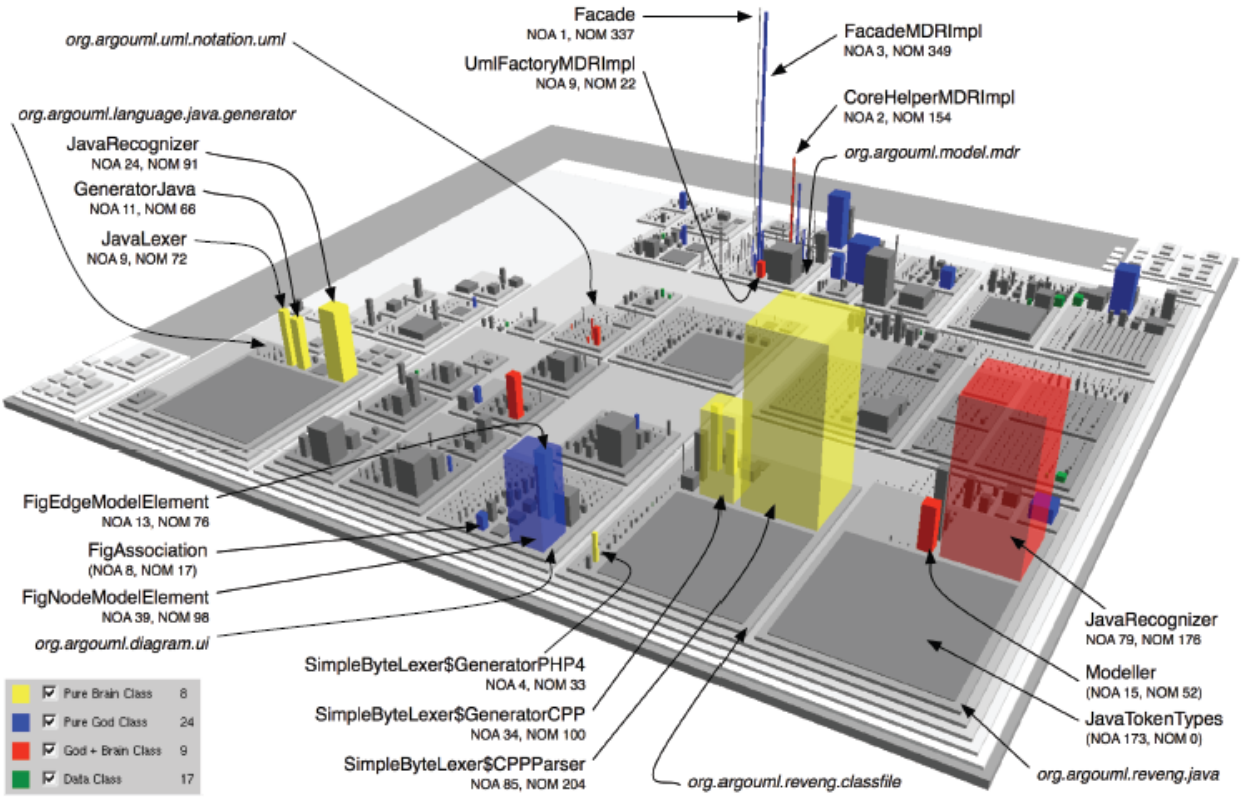


図 2-6： クラスレベル設計不調和（ArgoUML）

注意を引くのは，類似の形態を持つ３つのパッケージである．そのどれもが大きくて平坦な１つのビルと２、３の巨大なビルから構成される．

まず最初に org.argouml.revng.java の区画を見ていくと，この区画には巨大な赤いビル(JavaRecognizer)と小さな赤いビル(Modeller)があり，巨大な駐車場のように見えるビル(JavaTokenTypes)がある．

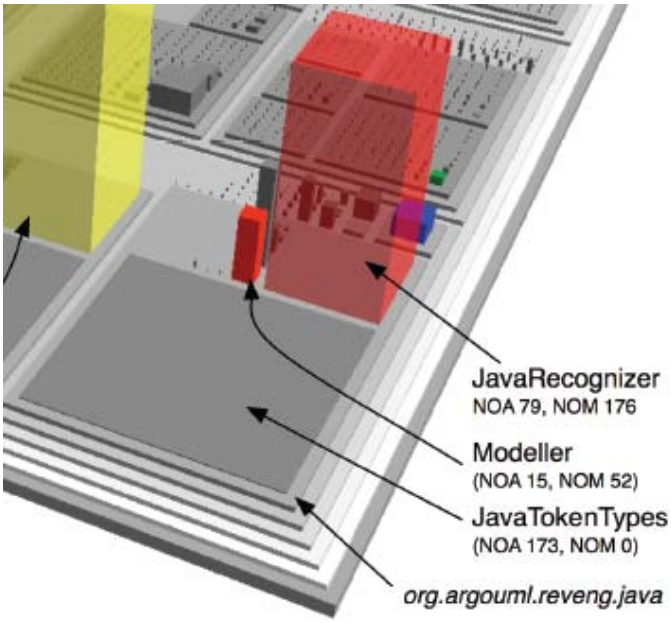


図 2-7: org.argouml.revng.java

二番目に見るのは org.argouml.reveng.classfile パッケージである．ここには二つのブレインクラスがある．一つは CPPParser であり，もう一つは GeneratorCPP である．

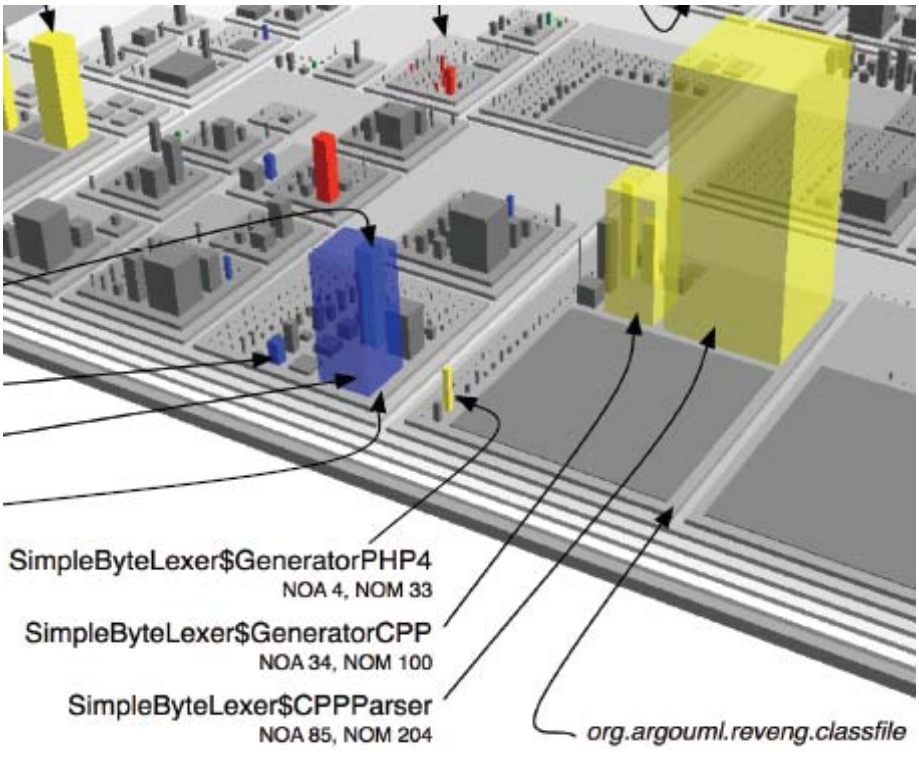


図 2-8: org.argouml.revng.classfile

三番目の類似パッケージは，org.argouml.language.java.generator であり、図の左側に位置している.このパッケージには，JavaRecognizer(24 アトリビュート,91 メソッド),GeneratorJava(11 アトリビュート，66 メソッド), JavaLexer(9 アトリビュート, 7 メソッド)の3つのブレインクラスがある.

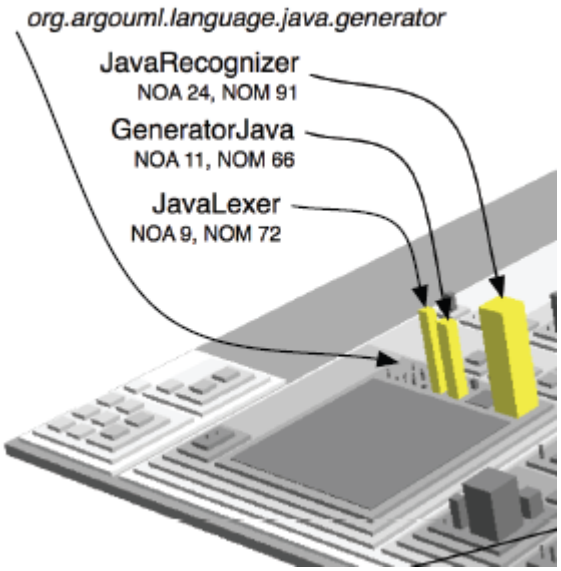


図 2-9: ArgoUML org.argouml.language.java.generator

2．6　メソッドレベル設計不調和

メソッドレベルの設計不調和を視覚化するために, よりきめ細かな表現方法を考案した. クラスレベル設計不調和では，クラスはレンガ状のブロックとして表現したが，メソッドレベルの設計不調和では，クラスは平板として表現する．そして、 メソッドをレンガ状のブロックとして表現し、クラスを表す平板の上に、メソッドを表すブロックを積み上げていく．
積み上げ方は，４つのブロックを１層とする．（図 2-10 右上の図を参照）

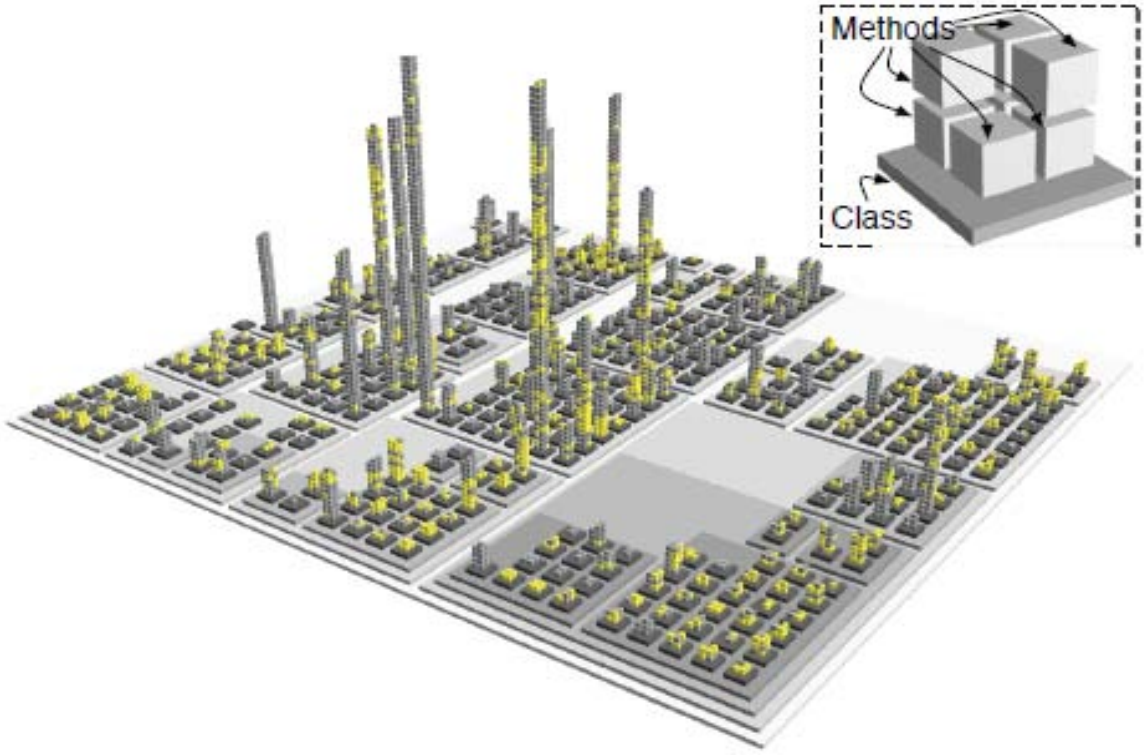


図 2-10：Jmol において、黄色で彩色した属性・操作の横恋慕と
ブロック表記したクラス詳細図(右上図)

ブロックの積上げ方式で Jmol を視覚化したところ，全メソッドの四分の一以上が属性・操作の横恋慕(Feature Envy)を示していた．リエンジニアリングを行い，この欠陥を除去することが望ましい。

図 2-11 は org.argouml.model パッケージを視覚化したものである．

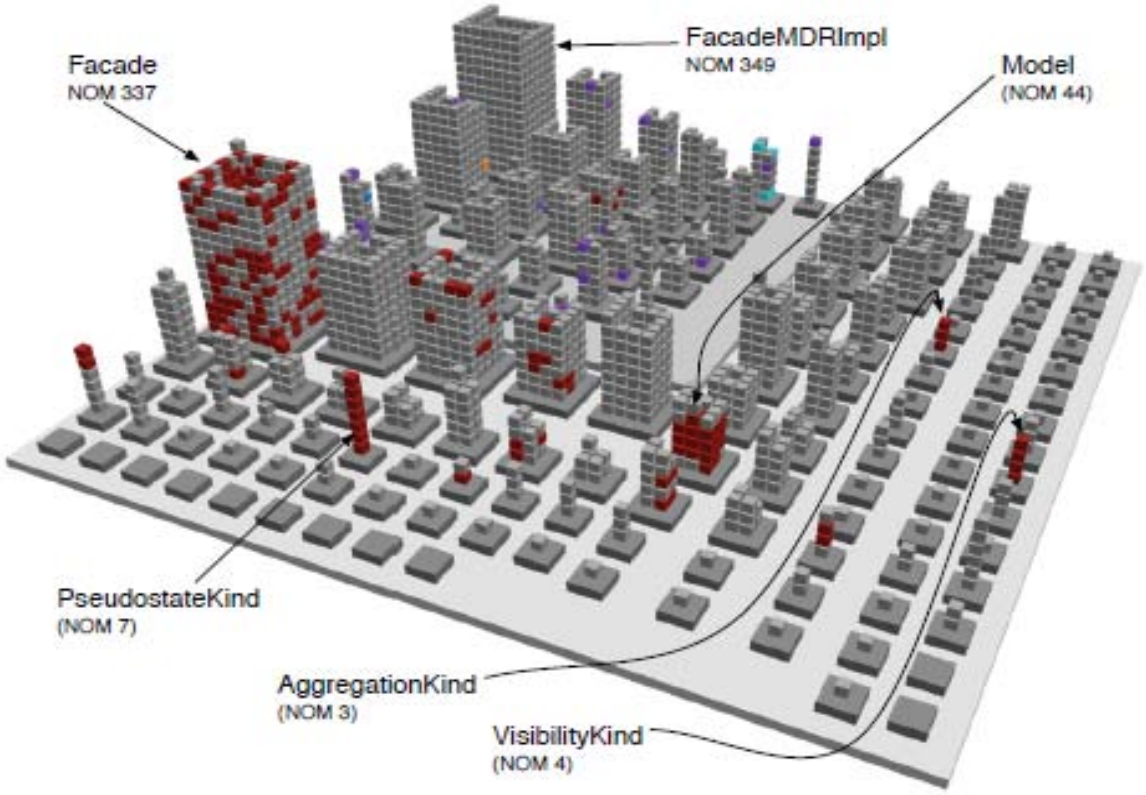


図 2-11： 赤く彩色した，変更の分散（org.argouml.model）

この視覚化には，プログレッシブブロック適合レイアウトを使用した．

メソッドレベルで表示させると，このパッケージを特徴付ける設計不調和は，「変更の分散(Shotgun Surgery)」であることがわかる．多くのビルに暗赤色のブロックがあるがこれが「変更の分散」に汚染されているメソッドである．更に暗赤色のブロックは reduced set of classes にだけ存在していることがわかる．この設計不調和が起こっているビルのうち最も大きなビルは，Facade インタフェースである

３．CodeCity の効果：ソフトウェア進化をたどる

3．1　はじめに

この章では主に，論文[6] の内容に基づき、CodeCity を使ってソフトウェア進化をたどる方法について説明する。（掲載した図も，論文[6]より引用．）

3．2　背景

ソフトウェア進化という現象が最初に研究対象となったのは 1970 年代,Lehman によってであった.その後,ソフトウェア進化の一連の法則として結実した.この法則はシステムが進化するにつれてシステムはより複雑になり,その結果,ソフトウェアシステムの構造を単純化し,元の状態に保つためにより多くのリソースが必要となるというものだった
　このことから,"変化は不可避であり,回避すべきではない"という概念に行き着いた.
過去十年でソフトウェア進化の研究は,CVS のような版管理システム～ほとんどがオープンソースコミュニティによるもの～に触発されてきた.
われわれはそのような（ソフトウェア進化の）研究の価値を強く信じている一方で,進化の現象そのものにはほとんど注目されていなかった事実を強調する.
例えば,「進化しつつあるシステムはどのように見えるのか？」という疑問に対する回答は，未だ得られていないが，通常提供されてきた，通時的なシステムのある特定の進化の側面を表わすラインチャート・バーチャートではその回答を得ることがほとんど期待できない．更にその上，この文脈において用いられる，code decay, design erosion, architectural drift などの多くの専門用語が受け入れられたことは認められるが，先の疑問が実際に暗示する心理的なモデルは未だ創り出されていない.

3．3　事例研究

CodeCity によるソフトウェア進化分析の事例として，論文[6]では３つのオープンソースソフトウェア ArgoUML, JHotDraw, Jmol を取り上げている.

System	ArgoUML	JHotDraw	Jmol
Packages	144	72	105
Classes	2,542	998	1,032
Lines of Code	137,000	30,000	85,000
Sampling Start	Oct 2002	Oct 2000	Jan 2000
Sampling End	Feb 2007	Apr 2005	Aug 2007
Sampling Period	random	weekly	8 weeks
Samples	9	57	50
Revisions	13,535	267	8,065

表 3-1： 調査対象システム

・ArgoUML は UML モデリングツールである
・JHotDraw は二次元グラフィックフレームワークである．彼らの論文では，一週間毎にソースコードのサンプリングを行なっている．
・Jmol は，分子化学構造の三次元視覚化ツールである。彼らの研究では，８週間に渡り，５０のサンプルバージョンを研究対象にしている．

3．4 時間を振り返る

ソフトウェア進化の視覚化は、表現の粒度と技法の二つにより性格づけられる．

表現の粒度には疎粒度(coarse-gained)と細粒度(fine-grained)の二つのレベルがあり，疎粒度レベルでは，クラスをれんが状のブロックで表現し，クラス内部の詳細を省略し，細粒度レベルでは，メソッドにフォーカスした表現とする．

進化の表現技法には三種類の技法がある．
１．世代マップ(Age map)　　世代分布を描くためのマップ
２．タイムトラベル(Time Travel)　システム履歴をステップ実行する
３．タイムライン(Time Line)　　分析対象のクラスやメソッドの全進化を単一のビューに捕える

この視覚化により，複雑で無形(intangible)のソフトウェア進化プロセスを有形化(tangible)視覚化し，システム進化についての洞察を得ることができる．

3.5 疎粒度表現

3．5．1 疎粒度の世代マップ

解説：システムのバージョンを示す彩色した都市をアーティファクトの世代に重ね合わせた．世代はバージョンを示す整数値であり，そのアーティファクトが生き残ったことを示す．彩色は，新規クラス・パッケージには淡い黄色を，古いものには暗い青を割り当てた．

目標：進化分析の開始点を確立する．システムの古い部分を見つける，最近変更された部分を見つける．時系列で振り返ることにより，システムの進化の全体的な感じを掴む．

適用：図 3-1 は ArgoUML0.24 の世代マップである。これはかなり大きなシステムでいくつかの巨大なクラスを持つ．

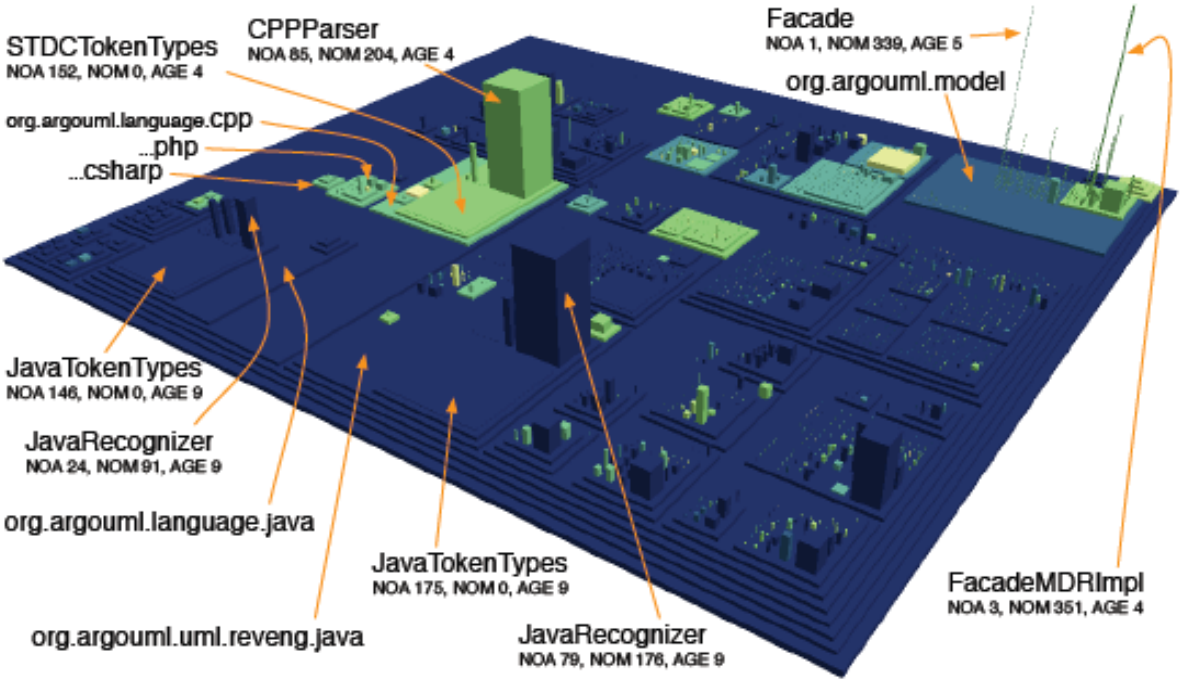


図 3-1： ArgoUML の祖粒度の世代マップ

JavaTokenTYpes と JavaRecognizer が org.argouml.language.java パッケージと org.argouml.uml.reveng.java パッケージの両方に表れている．世代マップによれば，両方ともシステムの歴史と同様に古いものである（両方とも暗い色で描かれている）この重複は，論文[1]における未解決の問題である．（論文[1]では，単一のバージョンのみ見ている．）

両方が初めからあったのだから，一方のクラスが他方に移動したか置換されたという仮説を捨てることができる．もう一つの発見は ArgoUML は Java 言語を最初の版からサポートしていたということである．C++, C#, PHP は後のステージになって追加された．それは org.argouml.language.cpp/php/csharp がそれぞれ黄緑色で示されていることによりわかる．

欠点：世代マップは、現在視覚化されたシステムの版について，進化の情報を実際よりもよく見せる．われわれが必要とするのは、進化プロセスそのものを視覚化する技法－例えば時間経過をたどれる技法－である．

３．５．２ 疎粒度のタイムトラベル

解説：タイムトラベルは，システムの履歴を時間順または逆順にステップ実行することにより，達成される．つまり，都市メタファにおいては，現在のバージョンを反映するために都市が自分自身を更新することにより達成される．場所(locality)が重要な役割を持っている：クラス・メソッド・パッケージなどの各アーティファクトが，都市メタファにおいては個別に土地(real estate)を割り当てられ，全進化を通じて維持されているということをわれわれは確認している．例えば、もしアーティファクトがシステムから削除されたりまたはある時点でシステムにはもはや存在しなくなった時，そのアーティファクトが占有していた場所は，他のアーティファクトによって占有されることのない空き地となる．都市進化のダイナミクス（例えば、ビルの成長、縮小、消失など）に際して、観測プロセスを容易にするために、われわれはカラートラッキングを使う．この機能は例えば、関心を持った実体(entities)に対して特定の色を手動で割り当てることができる機能である．これ(関心を持った実体に対する彩色操作）は全ての視覚化された版で維持される．

目標：全システムと個々のアーティファクト（パッケージ、クラスなど）の両方の進化を観測する．システムレベルでは、どの区画(districts)にヘビーメンテナンスがかかっているか、またはほとんど触られていないかを、関連する二つの版であるいは全進化において発見することを目標とする．都市内の特定のアーティファクトに焦点を当てることにより、その誕生を、選択したメトリクスの進化を、そしてあるケースにおいては死滅を、観測することを目標とする．

適用：図３において、われわれが見ることのできる一連のビューは，ArgoUML で抽出

された履歴についてタイムトラベルした結果得られたビューである．時間に関するよりよい感覚を提供するためにわれわれは図にリリース番号と日付をマークした．強調のために赤く色付けしたトラックは, 後の議論の対象となる要素である. 以前の実験(論文[1])からのもう一つの未解決の問題は，ArgoUML の進化の分析によって答えを見つけられれば，と考えている問題で，Facade インタフェースの興味深いケースである．その Facade インタフェースはひとつのクラスだけで３００以上のメソッドが宣言実装されている．ステップ実行によりわかったことは，明らかに不可解な設計上の決定に起因するものだった．バージョン 0.14 において，60 アトリビュート，１８０メソッドを持つ大きなビル ModelFacade クラスが現れた．そしてバージョン 0.16 において 108 アトリビュート 405 メソッドの巨大なビルに成長した．バージョン 0.18.1 で，ModelFacade は死滅した．しかし，それは二つの細長く高いビルの誕生と符号一致するような死滅だった．その二つのビルは，1 アトリビュート 306 メソッドを持つ Facade インタフェースと, 2 アトリビュート 319 メソッドを持つ NSUMLModelFacade 抽象クラスだった．
この変化の原因は可能性としては，ModelFacade が保守開発者(maintainer)の悪夢になるほどに成長しつつあることに開発者が気づいたか，あるいは，彼ら（開発者）が，その Facade の変種(variations)を定義する必要があったかである．彼ら開発者は一つのインタフェースで 306 メソッドを持つ共通ビヘイビア(common behavior)を宣言し，具象ビヘイビア (concrete behavior) を ModelFacade から新たなクラス NSUMLModelFacade に移動した．
バージョン 0.20 で第二の Facade 実装が生まれた．これは FacadeMDRImpl(2 アトリビュート,329 メソッド)と呼ばれた．これを観測するためにわれわれは, 少なくともインタフェースを表わす程度に高い(例えば NOM 値を参考にして)ビルを探した. 何故なら, Java ではインタフェースを実装するクラスは, インタフェース内で宣言された全てのメソッドを実装しなければならないからである．
バージョン 0.20 は二つのインプリメンターが共存する唯一の ArgoUML サンプルバージョンである．
バージョン 0.22 では NSUMLModelFacade クラスが消え，FacadeMDRImpl が唯一の Facade 実装として残された．

検証(Reality Check): ArgoUML の中心開発者(Key-developer)は，われわれのタイムトラベルで得られた洞察(insights)を認め，より詳細な情報を提供してくれた．
より高いレベルでは，MSUML から MDR へスイッチするモデルリポジトリは，より柔軟性が要求された．その柔軟性は，ModelFacade を共通インタフェースと具象クラスにリファクタリングすることによって実際に得られた．ModelFacade の多くの属性(attribute)は定数として実装されたトークンだった．Java1.4 ではエニューメレーション(enumerations)のサポートがなかったからである．

欠点：この粒度ではこの技法はクラス内部の進化についてはわからない．例えば，二つの比較対象バージョンの間で，追加した数と同数のメソッドを削除した場合，NOM メトリック値は変化せず，結果としてビルの高さは同じままとなる． このような詳細情報の消失は最後の 3 バージョンの ArgoUML(2006 年 8 月版と 2007 年 2 月版)とで見られる．

3．6　細粒度表現

より細粒度の表現の必要性を述べるために、われわれはメソッドをレンガ状のブロックに見立てて説明する．このブロックは４つのブロックを並べて一層にし，積み上げたものである（図 3-2 参照）
並べる順序は生成した順（古いものが下、新しいものが上）である。削除されたメソッドは空きスペースで示す。ビルの高さはメソッド数に対して均等であり続ける。ベースプラットフォームにはクラスへの直接アクセスを提供する．

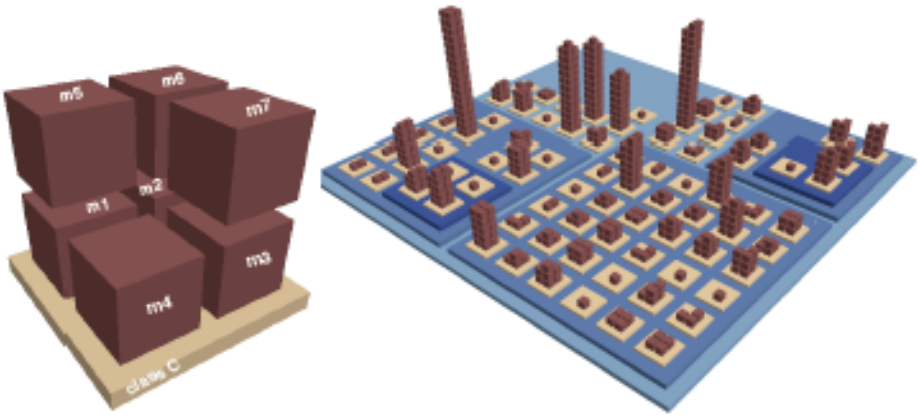
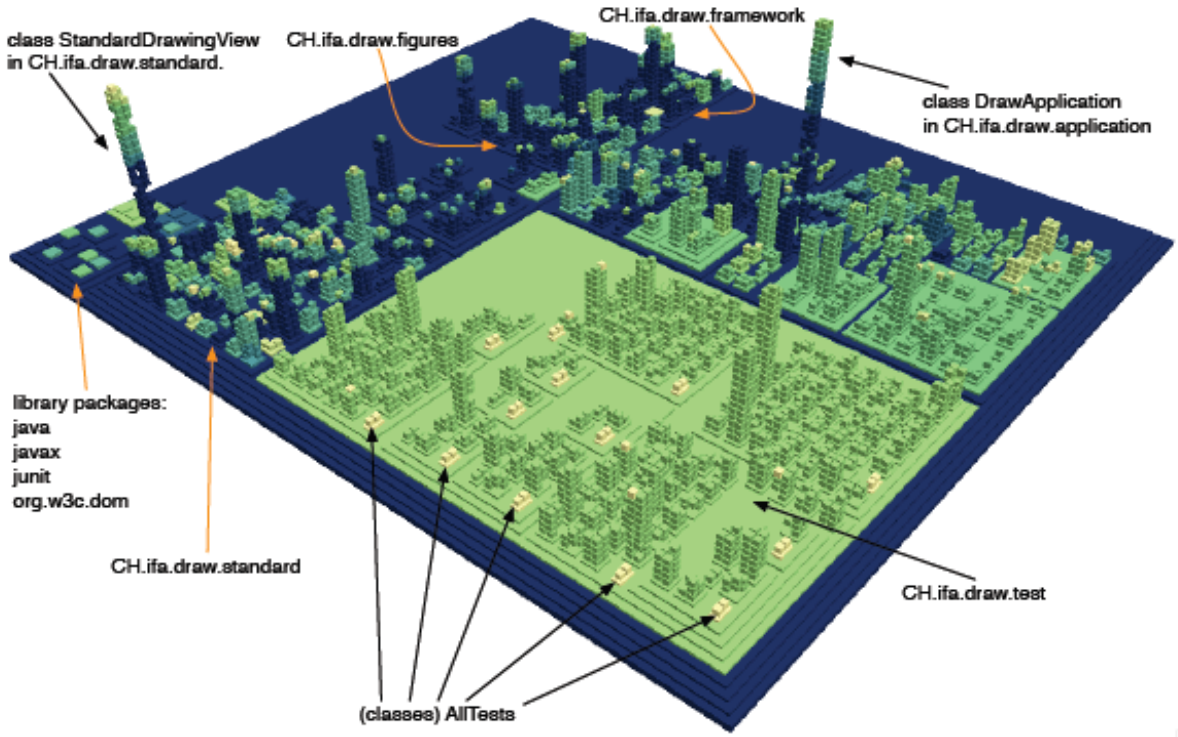


図 3-2：　細粒度表現（左図）と
ArgoUML の cognitive パッケージの細粒度表現(右図)



F　　図 3-3： 細粒度世代マップを JHotDraw の最終版に適用した CodeCity

３．６．１ 細粒度の世代マップとタイムトラベル

解説：前述の二つの技法が細粒度表現に適用される

目標：メソッドレベルの進化への知見を得ること．一回で生成されたクラス群とインクリメンタルに生成されたクラス群との対比を見つけること．

適用：細粒度世代マップは JHotDraw の最終版（図３－３参照）に適用され、この進化について興味深い結果が明らかになった．暗いシェードで色づけされた区画（例えば、CH.ifa.draw.standard/framework/figures)はシステム内でもっとも基底となるパッケージを表わしている．CH.ifa.draw.test パッケージは比較的新しく，8 サンプルバージョン中６バージョンに現れる．このパッケージは黄緑色で描かれる．淡い黄色の小さなビルも見える. これは AllTests クラスを表わし, バージョン７で追加されたものである. これらのクラスの各々には, main と suite という二つのメソッドが含まれる. main メソッドは, 全ての test サブパッケージに関するテストスイート(suite メソッドで定義される)を実行するための基点である．広範囲の色で塗られたビル(例えば，DrawApplication クラスと StandardDrawingView クラス)は最初のバージョンからシ

ステムの一部として存在する(底の部分の色が都市の地面の色と同じ)だけでなく，システム進化の間永続的に適合を要求されたクラスであることを示す（システムの各バージョン毎に、開発者達はメソッドを追加や削除をした)．　タイムトラベル技法を使用して test パッケージに注意を向けると，test パッケージは JHotDraw ライフタイムの終了に向けて全てのバージョンに一度だけ現れた．
最初にサンプリングした JHotDraw の履歴には３年半に渡り８バージョンにだけ含まれた.全てのユニットテストが突然現れたのは，2003 年 1 月から 7 月までの半年にかけてだった．この進化事実に関してより正確に理由づけをするために間隔を週単位にしてシステムをサンプリングした．そして驚いたことに結果は変わらなかった．test パッケージが現れたのはリビジョン１２１（2003 年 1 月 24 日）からリビジョン１５５（2003 年 1 月 31 日）までで、この 7 日間の間に３４コミットが実行された．
一度に全てのテストを書くとプロジェクトの後半でどうなるか知りたい．
(Writing all tests at once and at a late stage in a project is curious.)

検証(Reality Check): これらのクラスを作成した開発者がわれわれに語ったことによれば､彼は､未実装スケルトンのフォームを，将来実装された時のために使用し，JHotDraw のテストケースを自動生成するために JavaDoc ベースのコードジェネレータを使ったということだった．

欠点：あるメソッドが消えた場合、その理由はわからない．何故なら都市上には空き地しか現れないからである．

３．６．２ 細粒度タイムライン

解説：クラスバージョンはタイムラインに沿って互いに隣り合ったプラットフォームとして表現する．

タイムラインは左が最初のバージョン、右が最後のバージョンである．

メソッドはれんが状のブロックとして表現する．

この表現と世代マップ技法を組み合せて使い，世代（世代とは例えば、同じバージョンにおけるメソッドのグループなど）間の違いを，より明確に区別可能にする．

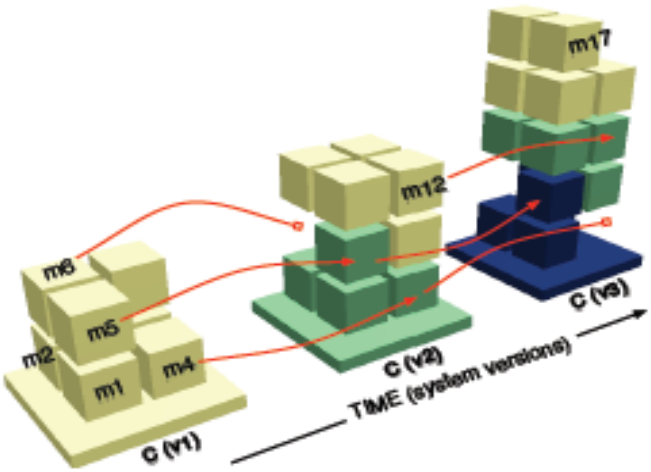


図 3-4： 細粒度タイムラインのマッピング原則

図３－４に例示したのは、上述の原則を，３バージョンの履歴を持つＣというクラスに適用したものである．最初のバージョンでクラスＣには m1 から m7 まで７メソッドがある．２番目のバージョンでは、m6 が削除され，m8 から m12 までの 5 メソッドがクラスに追加されている．追加されたクラスはビルのトップに現れており，残ったメソッドよりも明るい彩色がされている．
メソッド m6 があった場所は空き空間(empty space)として表わされている．
バージョン３では、古くからあるメソッド m4 が削除され，6 つのメソッドが追加されている．
この視覚化のメリットはアーティファクトの進化を完全に表現できることであり，したがって，進化に関連したパターンの発見につながる．

目標：削減されたアーティファクトセットを孤立させ，完全な履歴を表現するビューを生成する．完全な履歴には、全てのインナーコンポーネントが含まれる。そして、インクリメンタルに成長するクラス，循環(recurring)メソッドなどの進化のパターンを調査する．

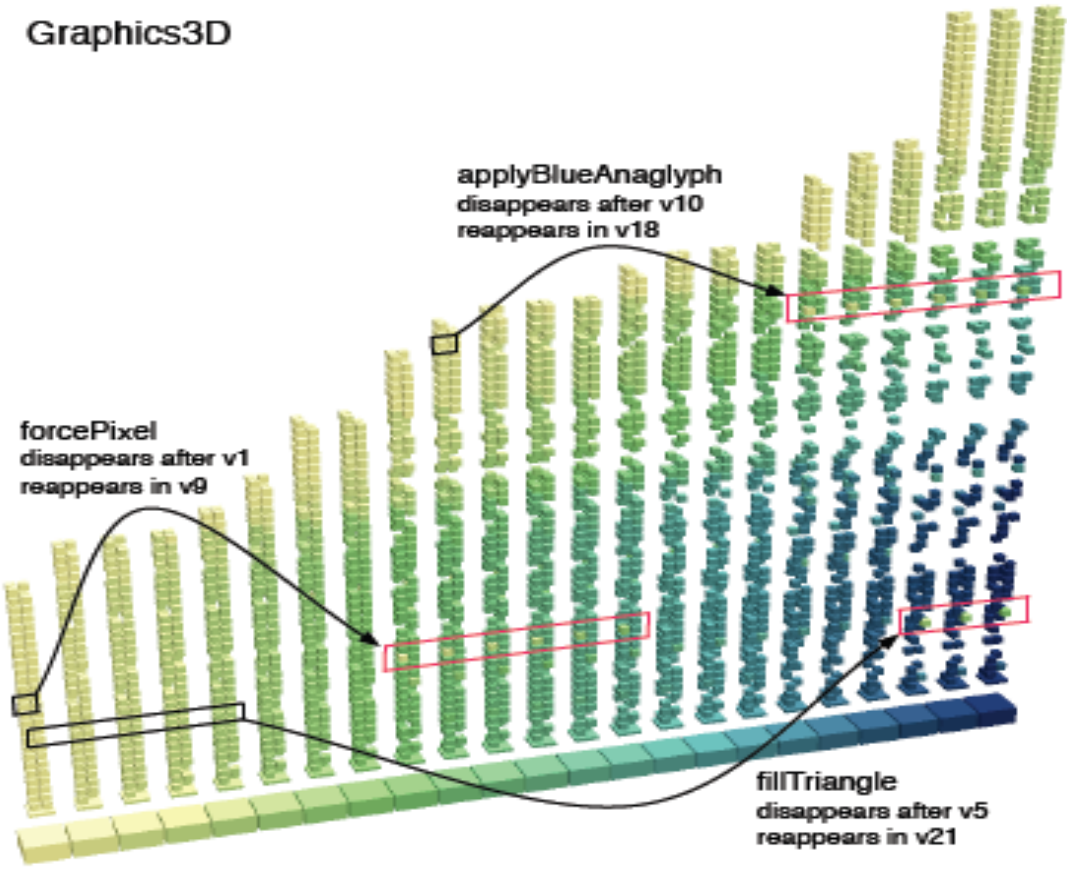


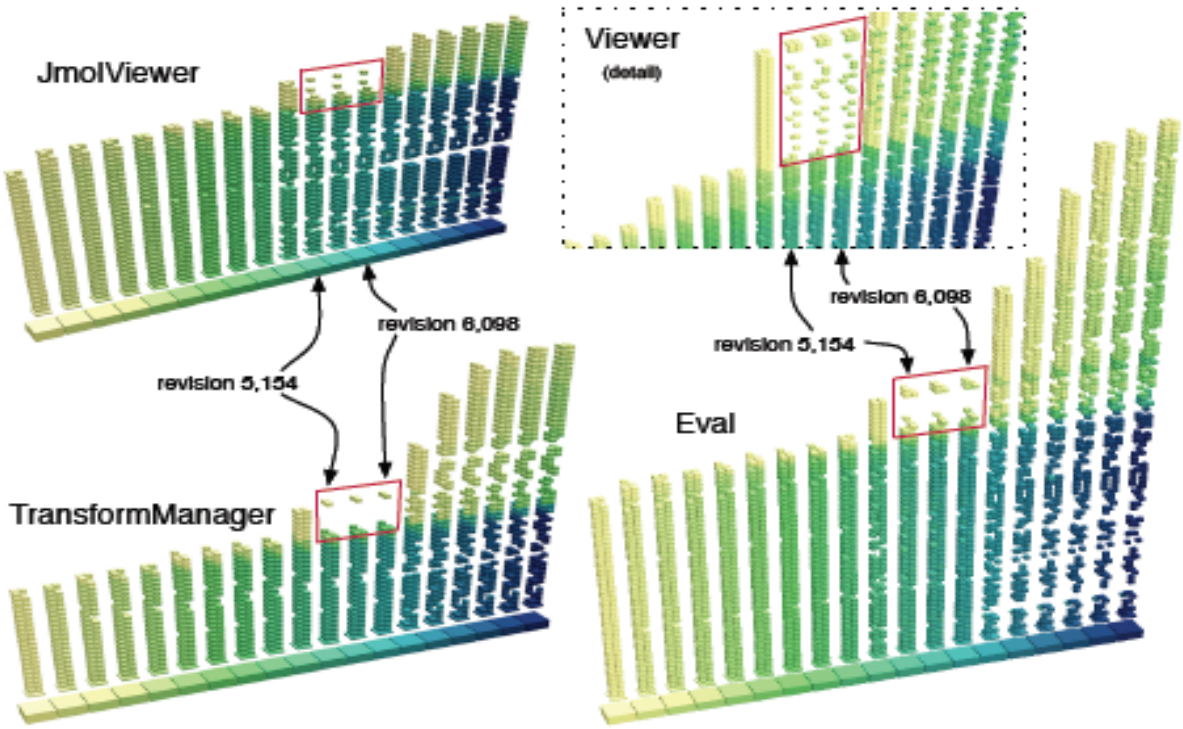
図 3-5： Graphics3D クラスのタイムライン

適用：　われわれはタイムラインを用いて多くの Jmol クラスを分析した．
最初の例は、Graphics3D クラスである．このクラスは、システムの５０バージョンという長いサンプル履歴中２３バージョンに現れた．（図 3-5 参照）
Graphics3D はおそらくコアクラスである．何故なら、最初のバージョン以来それは大変多くの(103)メソッドに依存してきた．衰退の兆候は９番目のサンプルバージョンに現れる、そしてそれ以後の後続バージョンにおいて衰退の傾向が強くなる．同時に新しい機能がこのクラスに次第に追加される．最終バージョンは継続的な適合の歴史を反映している。３１１メソッド中１５８メソッドが最終バージョンで作成された．

図 3-5 では興味深いパターンも例示する．無くなったブロックが後で復活する場合がある。これは削除されたメソッドが後で復活することを意味する．このパターンは特にわれわれの技法では発見が容易である．世代マップ技法の成果とレイアウト上にインポーズされた通時的な順序は目につきやすいからである。メソッド復活後、復活したメソッドを表わすブロックは、変種として目立つ色になっている．この彩色によって，メソッド列のカラーパターンの変化が滑らかではなくなっている．

この進化パターンを更に詳細に見るために、4 クラスの履歴をタイムライン上に並べてみた．その図を図 3-6 として示す．

Ｅｖａｌはその進化の過程で多くのメソッドを失っている（432 メソッド中 166 メソッド）その結果、最終バージョンでは多くのブロックが欠けており不安定に見える。
ソフトウェアの衰退を大変示唆に富んだ方法で描写できることが細粒度タイムライン技法では可能である．



Class History	Total Methods	Current Methods	Removed Methods
JmolViewer	177	150	27
Viewer	1029	750	279
TransformerManager	220	161	59
Eval	432	266	166

図 3-6： Jmol における類似した 4 パターンのクラス

図 3-6 に示すのは，Viewer のタイムライン詳細と，JmolViewer, Eval, TransformManager 3 つのクラスの全タイムラインである．Viewer クラスには全履歴を通じて 1029 メソッドがあった．
これらのタイムラインを見ると，2006 年 5 月 22 日のリビジョン 5154 で削除されたブロックが 2006 年 11 月 6 日のリビジョン 6098 で復活している．
われわれの仮説は次のようなものである。開発者達は論理的な結合クラスからメソッドを削除し、それが不具合の元となったが、その不具合はすぐには見つからず、後に削除

したメソッドを復活させることにより、修正した.

検証 : 2006 年 5 月 10 日のリビジョン 5091 のログには,「g3d シェイプの描画ルーチンに javax.vecmath.Point3f などもういらない. かつては、画面軸座標を Point3f オブジェクトとして渡さなければならないケースがあったが、...」とあった.
そして, 2006 年 9 月 17 日のリビジョン 5579 のログによれば, 以前削除したメソッドの復活をしぶしぶ認めていた.（ログの内容は「vecmath ライブラリを前の状態に戻す変更」とあった.）
また、3 人の Jmol 開発者がこれを認め、グラフィックライブラリにパフォーマンス上の問題があり、レンダリングが遅かったため、いくらかのリファクタリングを施した後、以前の状態に戻した、と話してくれた.

欠点 : リビジョンが数百のレベル以上に及ぶとスケーラビリティの問題がある.

3.7 結論

われわれが提案した技法は，オブジェクト指向システムの進化を調べるための三次元都市メタファに基づく視覚化技法である．われわれの技法を，バージョン情報を持つシステムに適用すると，そのシステムの進化プロセスをよりよく理解できる．そして、過去のバージョンを個別に取り出してきただけでは見出せないような情報を明らかにする（ただし，あくまでも履歴をたどって判る限りの情報である.）

４． References

[1] R. Wettel and M. Lanza. Program comprehension through software habitability. In Proceedings of ICPC 2007 (15th International Conference on Program Comprehension), 2007

[2] R. Wettel and M. Lanza. Visualizing software systems as cities. In Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis), 2007

[3] R. Wettel and M. Lanza. CodeCity: 3D Visualization of large-scale software. In companion Proceedings of ICSE 2008 (30th International Conference on Software Engineering), Research Demonstration Track, pp. 921 - 922, ACM Press, 2008.

[4] R. Wettel and M. Lanza. CodeCity. In Proceedings of WASDeTT 2008 (1st International Workshop on Advanced Software Development Tools and Techniques). 2008

[5] R. Wettel and M. Lanza, Visually localizing design problems with disharmony maps, In Proceedings of Softvis 2008 (4th International ACM Symposium on Software Visualization), 2008

[6] R. Wettel and M. Lanza. Visual exploration of large-scale system evolution. In Proceedings of WCRE 2008 (15th Working Conference on Reverse Engineering, 2008

[7] R. Wettel, Scripting 3D visualizations with CODECITY, In Proceedings of FAMOOSr 2008 (2nd Workshop on FAMIX and Moose in Reengineering), 2008

2．2 研究2:ソフトウェア視覚化の歴史

はじめに

研究 2 は、Richard Wettel の 2010 年の論文「Software System as Cities」の Chapter2 の翻訳である． この章を読むことにより, CodeCity 以前の Software visualization の研究状況、及び、CodeCity が登場する背景がよりよく理解できるとわれわれ研究グループでは考えた． これが、Chapter2 の翻訳をここに載せる理由である． 翻訳までの経緯であるが、この章の翻訳の提案を石川が行ない、実際の翻訳は小林が行なった． この章の理解には、原文に収められている図表が必要と考えられたため、峯村が、原文から図表をスキャンし、本文への取り込みを行なった． 更に石川がその図を PNG フォーマットに変換した． 翻訳された文書については B グループメンバーがレビューに加わった．ただし，レビュー結果は微調整に留まった．なお，翻訳の図表番号は原文通りとした．

この文書の原文は、以下の URL で入手できる。

http://www.inf.usi.ch/phd/wettel/download.php?f=Wettel10b-PhDThesis.pdf

Chapter 2
A History of Software Visualization（ソフトウェア視覚化の歴史）

Price他はソフトウェア視覚化（visualization）を次のように定義している。

> “コンピューターソフトウェアに対する人間の理解とその効果的な利用を容易にするために、活版印刷術（typography）、グラフィックデザイン、アニメーション、映画撮影技術などの技能を、近代的な人間－コンピューターの相互作用（インタラクション）やコンピューターグラフィックスの技術と合わせて利用することである”。

Diehlによると、ソフトウェア視覚化はソフトウェアの構造、行動、進化を視覚化することに関わっている。構造の視覚化はシステムの静的な部品やその関係に焦点を当てており、プログラムを走らせることなくソースコードから直接に抽出することが出来る。行動の視覚化はプログラムの実行から抽出できる動的な情報に焦点を当てる。進化の視覚化はソフトウェアシステムがその生涯（lifetime）を通じて影響される変化に焦点を当てる。この文脈で言うと我々の研究はソフトウェアシステムの構造と進化に焦点を当てている。

今日では、ソフトウェア視覚化は確立されたソフトウェアエンジニアリングの分野であり、いくつもの専門の立場とそれを取り巻く盛んなコミュニティがある。しかしこの状態になるまでに50年かかっている。この章では進歩のきっかけとなった要因という視点でソフトウェア視覚化への重要な貢献について述べる。

2.1 Foundations of Visualization（視覚化の芽生え）

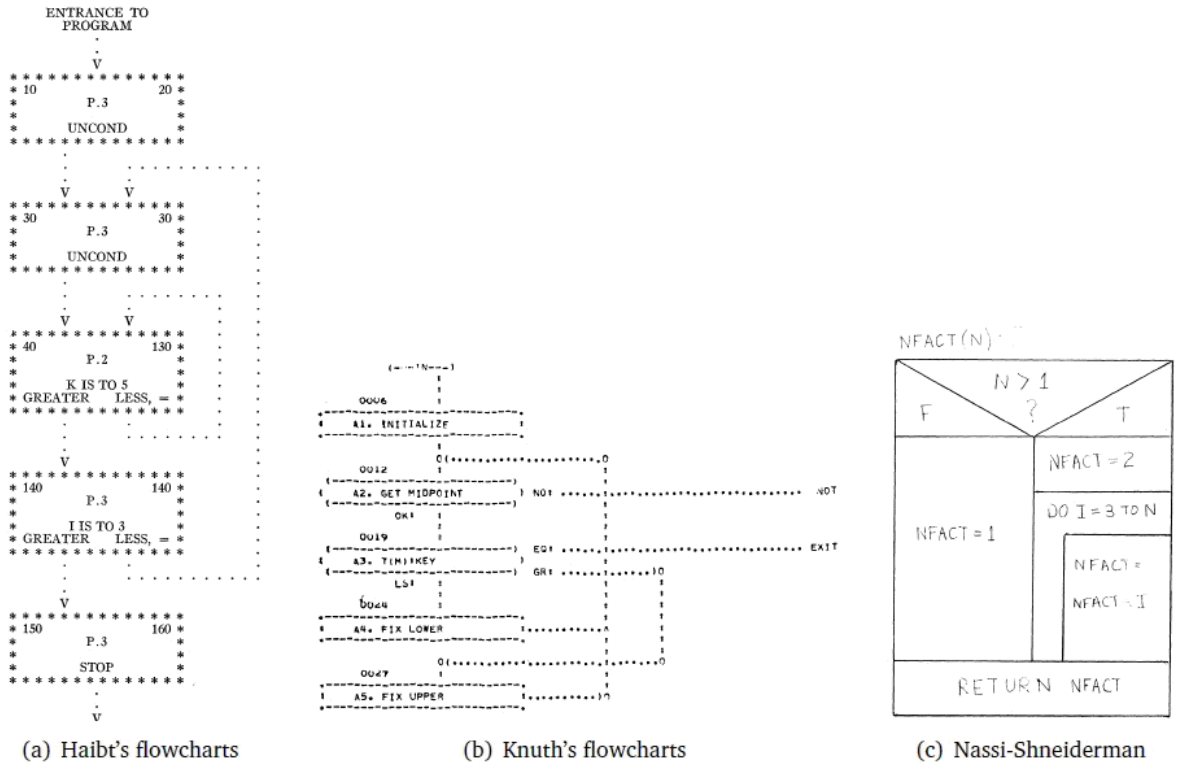
事実（facts）を理解するために視覚（vision）を利用するということに対する関心は、ソフトウェアが出現するよりずっと以前に始まっていた。最初のデータグラフは1786年

にスコットランドの技術者Playfairによって公表された。彼は統計学におけるグラフ手法の創始者であるとみなされており、折れ線グラフ、バーチャート、パイチャートを発明した。

情報の視覚化分野についての最初の理論的な基礎は 1967 年に Bertin の注目すべき“semiology of graphics”（グラフィックスの記号学）[Ber67]によって築かれた。この中でこのフランスの地図製作者はダイアグラム（diagram、図）を設計するときの枠組み（フレームワーク）を述べている。

2.2　Pre-1980s（1980 年以前）

ソフトウェア視覚化の最初の形はダイアグラム（diagram、図）であった。プログラムの包括性（comprehension）を表すのに役立つとみなされて、ダイアグラムに対して非常に大きな研究努力がはらわれた。1959 年に Haibt はアセンブラー言語や Fortran で書かれたプログラムに基づいてフローチャートを描くシステムを作りだした（Fig2.1(a)）。1963 年に Knuth は文書化の道具としてフローチャートを作りだすことのできるシステムを開発した。より構造的な Nassi-Schneiderman ダイアグラムはフローチャートに替わるものとして 1973 年に現れた。



(a) Haibt's flowcharts　　(b) Knuth's flowcharts　　(c) Nassi-Shneiderman

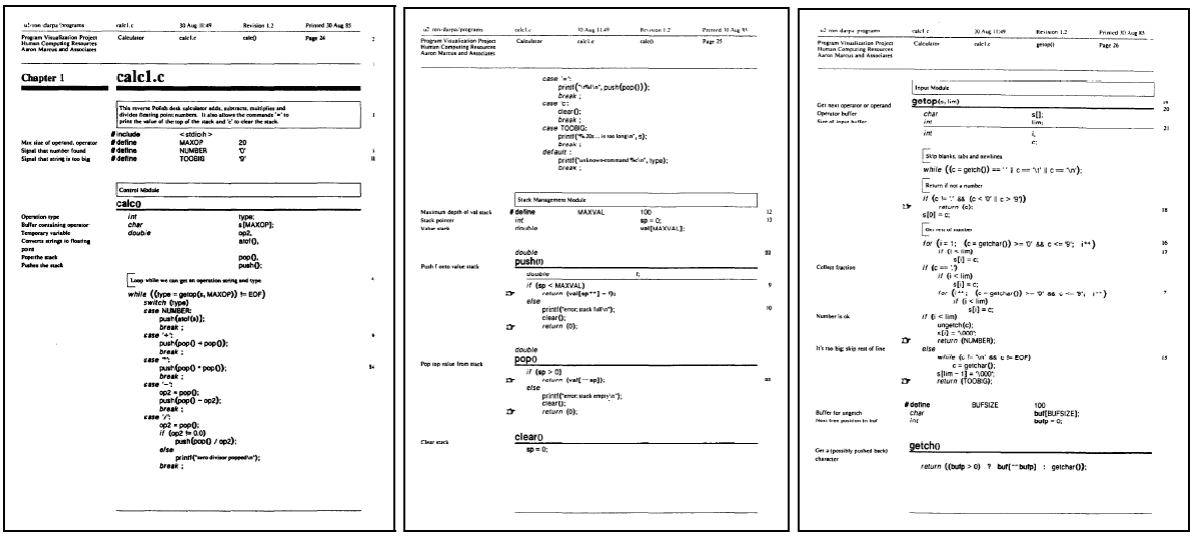
<<Figure 2.1. The first generation of diagrams>>

やや原始的な形のソフトウェア視覚化はソースコードの表現、もっと読みやすいプログラムの書き方に対する要求であった。もっとも古いソースコードの表現は pretty-printing であった。そこではコードの読みやすさを高めるために、行送り、インデンテーション、配置（layout）などを用いていた。最初の pretty-printing は 1970 年代に現れたが、それらは、組み込みの pretty-printer を含んでいた LISP、PL/I、Pascal などいくつかのプログラミング言語に専用のものであった。

近代視覚化のもう一つのさきがけは、プログラムの振舞いのアニメーションであり主に教育の目的で使われた。1966 年に始まる Knowlton の 2 つのフィルム（Bell Laboratories で開発された低レベルプログラミング言語について説明したもの）は、プログラムの振舞いを説明するためにアニメーション技術を初めて適用した代表的な例である。

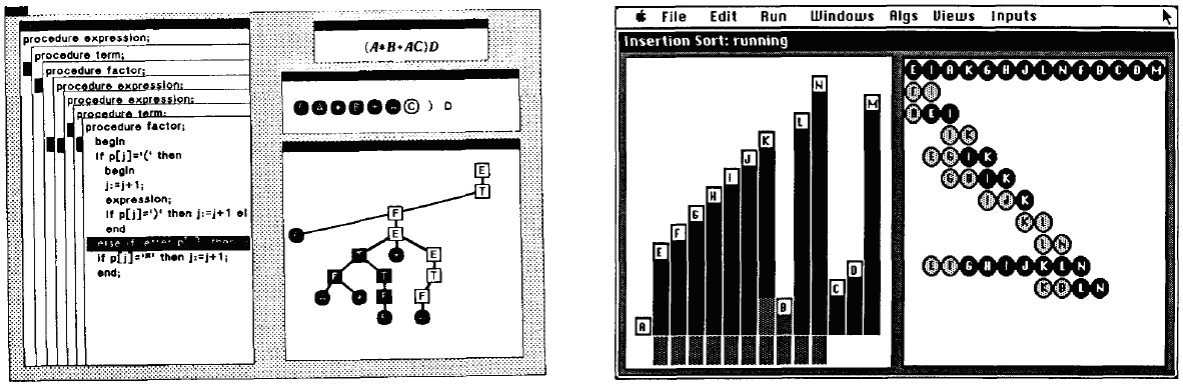
2.3 The 1980s（1980年代）

Pretty-printing の次のステップは Xerox Cedar community で採用されたような、書体、数式表記、ヘディングなどを使用できるようにすることであった。同じような発想で Knuth の WEB system は、マークアップ言語を使うことによってソースコードと文書を結合し一つの出版物（publication）とした。他方、Baecker と Marcus による SEE Program Visualizer はグラフィックデザイン規範（graphic design principle）にもとづくスタイルガイドによって C プログラムを製版することができ、一群の C プログラムから“program book”を作ることができた。



<<Figure 2.2. Excerpt from a program book produced with the SEE Program Visualizer>>

1980 年代に中心的に研究されたソフトウェア視覚化の目標はおそらくプログラムの振舞いであり、主に教育の目的で使われた。プログラムの振舞いアニメーションの中で最もポピュラーな早期の例は 1981 年から始まる Baecker の“Sorting Out Sorting”のフィルムで、shell-sort、 bubble-sort、 quick-sort など最も重要な分類アルゴリズム（sorting algorithm）のいくつかを理解するための視覚的な支援を提供するものであった。この時代の最も突出した視覚化システムは、アルゴリズムをアニメ化して教育的な支援として使われた Balsa とその直接の後継者 Balsa-II である。（Fig 2.3）



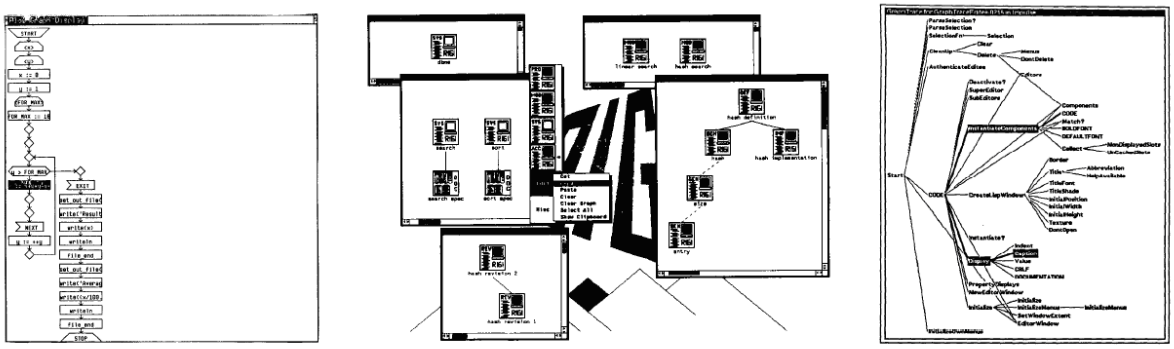
<<Figure 2.3. The Balsa(left) and Balsa-II (right) algorithm visualization systems>>

1983年にMyersはXerox Palo Alto Research Center on Incenseにおける彼の成果、即ちレコード、ポインター等のデータ構造を視覚化するためのシステムを公表した。

ディスプレイを装備したパーソナルコンピューターが使えるようになって、研究者は広範な聴衆に対してインタラクティブなソフトウェア視覚化システムを作りだすことが出来るようになった。Reissは早期の統合開発環境（IDE：Integrated Development Environment)で、1984年以降Pecanと呼ばれる単純な視覚化を含む複数viewの利用を試みている。（Fig 2.4(a)）

1986年にMüller他はRigiを提示した。ソフトウェアシステムの構成要素とその関係によってソフトウェアシステムの構造を視覚化することを目的としたシステムである。長期間にわたって様々に使われるという試練に耐えて、最もポピュラーな早期のソフトウェア視覚化システムとなった。（Fig 2.4（b））

主流としてオブジェクト指向プログラミング言語の数が増え続けることによって、単に構造を理解するだけでなくこのパラダイムによって書かれたシステムの行動を理解することに対する関心が高まった。Kleyn 他は GraphTrace を提案した。動的な情報を視覚化するために並列的にアニメ化された視覚（view）を利用したツールである。（F ig 2.4（c））



(a) Pecan　　(b) Rigi　　(c) GraphTrace

<<Figure 2.4. Examples of prominent systems from the 1980s>>

様々な型の視覚化が発生することによってそれらを分類（categorize）し、体系化する必要が出てきた。この方向の最初の試みはMyersのプログラム視覚化システムの分類法（taxonomy）である。Myersはプログラム視覚化システムを2つの軸に沿ってクラス分けした。第1軸はプログラムのどの部分（コード、データ、アルゴリズム）が視覚化されているかを示す。第2軸は表示される情報が静的であるか動的であるかということを示す。

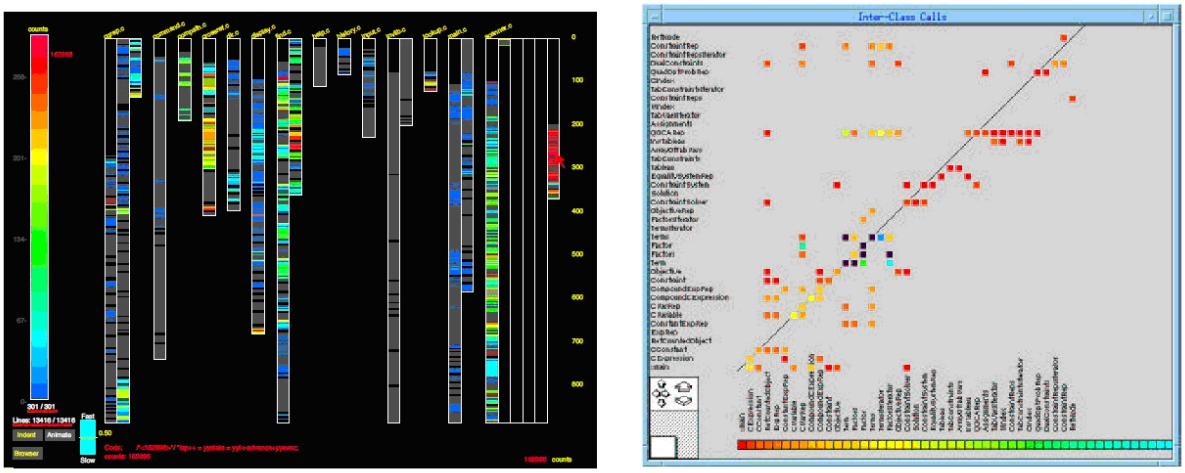
数年後、Price 他はもっと詳細な分類法を提案した。それによるとソフトウェア視覚化は 6 次元に基づいて分類される：目的（scope）、内容（content）、形式（form）、方式（method）、相互作用（interaction）、効果性（effectiveness）である。これらの分類軸間の直交性が相対的である（relative orthogonality）ということによって、ソフトウェア視覚化を分類することの困難さが顕在化した。

2.4　The 1990s（1990年代）

時間が経つにつれて、ますます多くの研究者がソフトウェア視覚化に興味を持つようになり、今までに無かったこの新しい研究分野について調査を始めた。結果としてソフトウェア視覚化時代に可能性を持ついくつかの成果がこの時期に生まれた。1992 年 Eick 他は SeeSoft を提示した。ソフトウェアシステムの非常に細かく粒度化された（コードの各行レベルにまで細分化された）視覚化を提供することが出来るソフトウェア視覚化ツールである。

その後、Ball と Eick はいくつもの場面（context）に SeeSoft を適用することに成功した。それぞれの場面は、例えば静的な属性、効率のプロファイル（Fig 2.5(a)）、バージョン履歴（すなわち、進化）などソフトウェアの異なった側面を表していた。

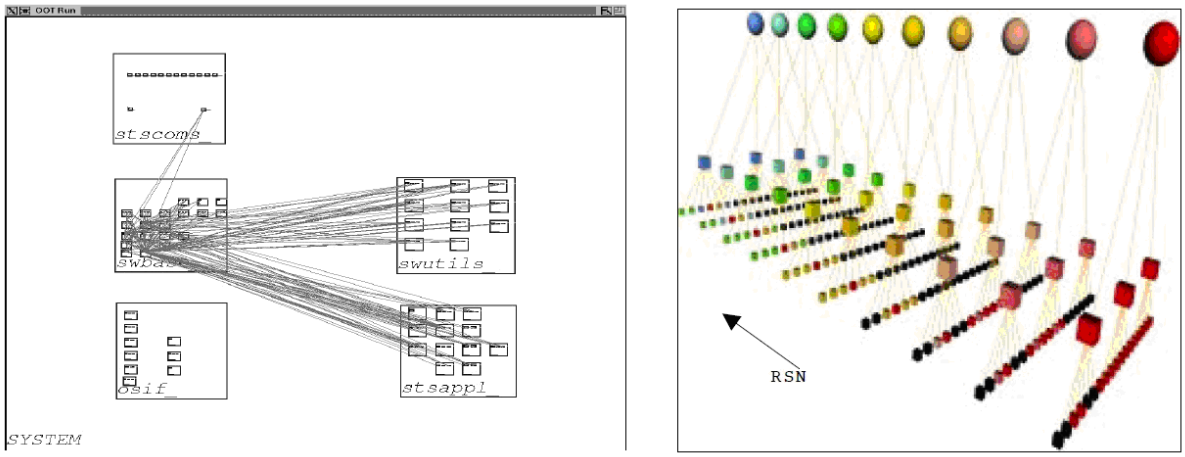
1993 年に、プログラムの動的な視覚化の場面で、De Pauw 他は、オブジェクト指向システムの振舞いについて今までにない新しい視点を持ったツールである Jinsight（Fig 2.5(b)）を導いた。



(a) Program execution in SeeSoft　　(b) Inter-class call matrix in Jinsight

<<Figure 2.5. Examples of dynamic visualization >>

ソフトウェア進化の視覚化に対する最初の貢献者は 1996 年の Holt と Pak である。彼らは産業ソフトウェアシステムの 11 バージョンについて、GASE（Fig 2.6(a)）と呼ばれる彼らのツールを利用して構成的な（architectural、アーキテクチャ的な）データを視覚化した。3 年後、Gall 他は別の産業システムの進化をそのリリース歴に基づいて分析するのに、従来の 2D 視覚化と新しい 3D（Fig 2.6(b)）視覚化の両方を利用した。



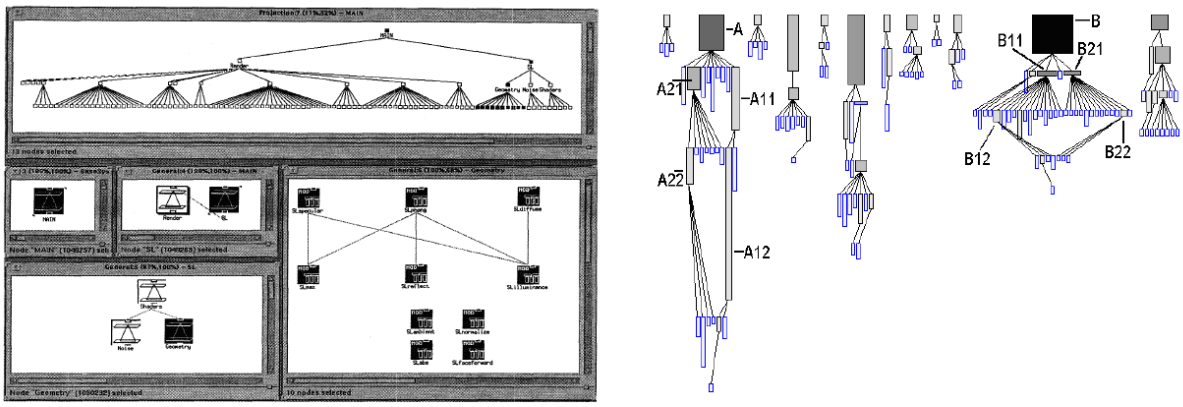
(a) Changing dependencies in GASE　　(b) 3D Visualization of release histories

<<Figure 2.6. Early visualization of software evolution >>

1995 年に Storey 他は SHriMP を提示した。ソフトウェア構造の視覚化ツールであり（Fig 2.7(a)）、複数の視野を持つ（multi-perspective）ことと魚眼（fisheye）技法によって Rigi 環境を強化して得られたものである。

1999 年に Lanza は、CodeCrawler ツールの中に実装された polymetric views と呼ば

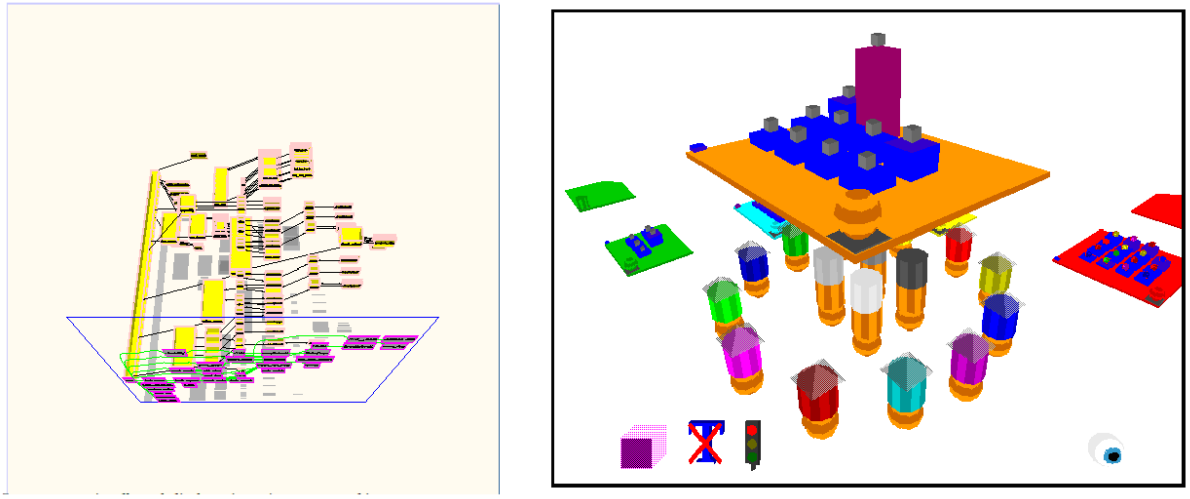
れるソフトウェアシステムの軽量視覚化を提案した。Polymetric views は、ソフトウェアメトリックスで強化された、簡潔でありながら効果的な視覚化であり、オブジェクト指向ソフトウェアシステムの分離状態（outlier）やパターンを明らかにすることができた。



(a) Fisheye views of nested graphs in SHriMP　　(b) Polymetric views in CodeCrawler

<<Figure 2.7. Examples of static visualization of system structure>>

この期間は、3D（3 次元の）視覚化やバーチャルリアリティ（VR）のように今までに無かった新しい研究の方向を実験するのに好適な時期でもあった。1995 年に Reiss は、プログラムの 3D 視覚化を構築する PLUM と呼ばれる配置可能（configurable）なエンジンを提示した。（Fig 2.8(a)）
1998 年に Young と Munro は VR の中でプログラム理解（comprehension）のためのソフトウェアの表現を追求した。その例を Fig 2.8(b)に示す。



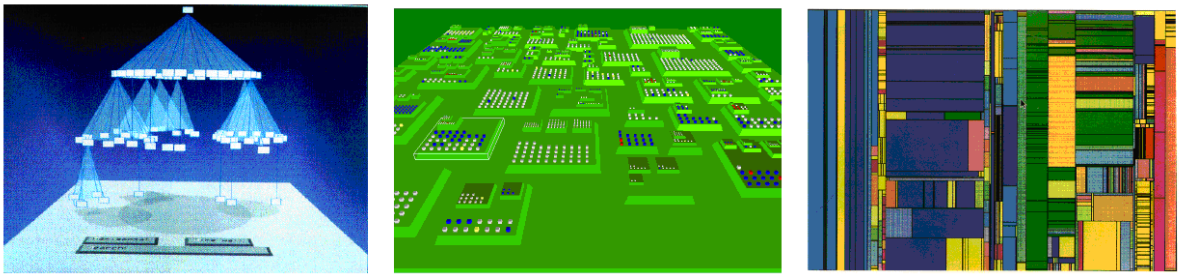
(a) PLUM　　(b) Software visualized in a VR environment

<<Figure 2.8. Early examples of 3D software visualization >>

1990 年代に行われた大量の、多岐にわたる貢献の結果、ソフトウェア視覚化分野が強い勢い（momentum）を持つことになった。

より広い範囲を対象とする情報視覚化の分野がソフトウェア視覚化研究への刺激（inspiration）の源泉となっていた。まず情報を効率よく表現するために視覚的な手段を使うということについての Edward Tufte の素晴らしい成果が、ソフトウェア視覚化を含む情報視覚化全体に影響を与えた。

さらに、いくつもの情報視覚化技術がソフトウェア視覚化技術に刺激を与えた。Robertson 他による Cone Trees (Fig 2.9(a))、Andrews 他による Information Pyramids (Fig 2.9(b))、Shneiderman によって考案された tree-maps (Fig 2.9(c))などである。



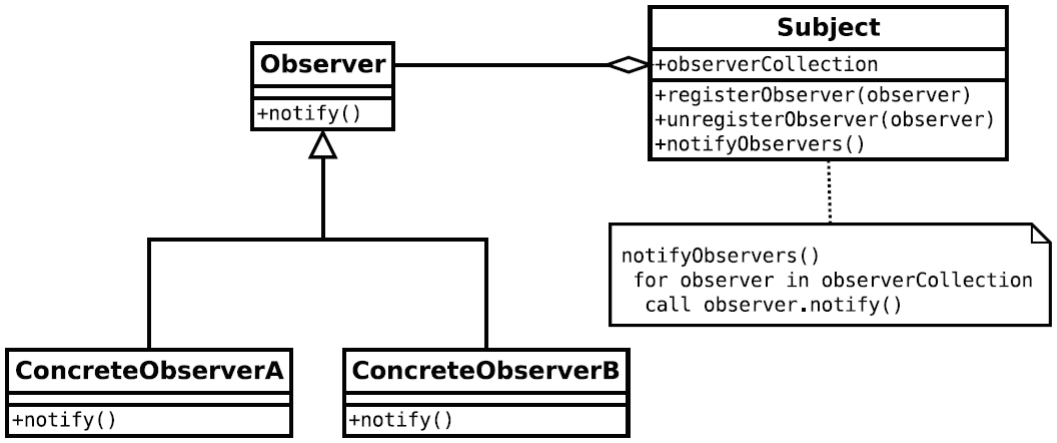
(a) Cone Trees　(b) Information Pyramids　(c) Tree-map

<<Figure 2.9. Information visualization techniques that inspired software visualization>>

UML の誕生

視覚化アプローチではないが、UML（Unified Modeling Language）はソフトウェア視覚化と密接に関連している。UML はダイアグラムを基本とした言語であり、ソフトウェアの静的な側面（例えば、クラス図、パッケージ図、コンポーネント図など）と動的な側面（例えばシークエンス図、タイミング図など）の両方を書き表すために使われる。Unified Software Development Process の一部として UML は現在の業界におけるモデリングのデファクト標準である。

UML の最もポピュラーな利用は、有名なデザインパターン（ソフトウェアデザインにおいて共通に発生する問題に対して一般的に再利用可能な解である）を記述するときに視覚的な支援を行うことであった。Fig 2.10 は *Observer* デザインパターンに対応するクラス図を示している。



<<Figure 2.10. A UML class diagram >>

2.5　The 21st Century（21世紀）

21 世紀の初めまでに、情報視覚化は多くの専門的な立場が主張される確立された研究分野になっていた。それに対して、ソフトウェア視覚化はプログラム理解やリバースエンジニアリングの問題を想定した、限られた人達に特有な手段であるとみなされていた。

2001 年に Dagstuhl で、ソフトウェア視覚化の領域で活動するもっとも著名な実務者や研究者を集めて“ソフトウェア視覚化に関する国際セミナー（International Seminar on Software visualization）”が開かれた。世界中から集まった 50 名の研究者がソフトウェア視覚化の最新の技術レベルについて議論を行い、この分野の将来へ向かっての課題を確認した。さらに重要なことは、彼らがソフトウェア視覚化に関する一連の国際的な議論を立ち上げることを決定したことである。

その後、ソフトウェア視覚化を専門とする 2 つの議論の場が創始された。2002 年に始まった IEEE の VISSOFT（International Workshop on Visualizing Software for Understanding and Analysis）と、それに続いて 2003 年に行われた ACM の SoftVis（Symposium on Software visualization）である。この時期にソフトウェア視覚化分野によって得られた力（momentum）によって、その後ソフトウェア視覚化アプローチが爆発的に広がった。

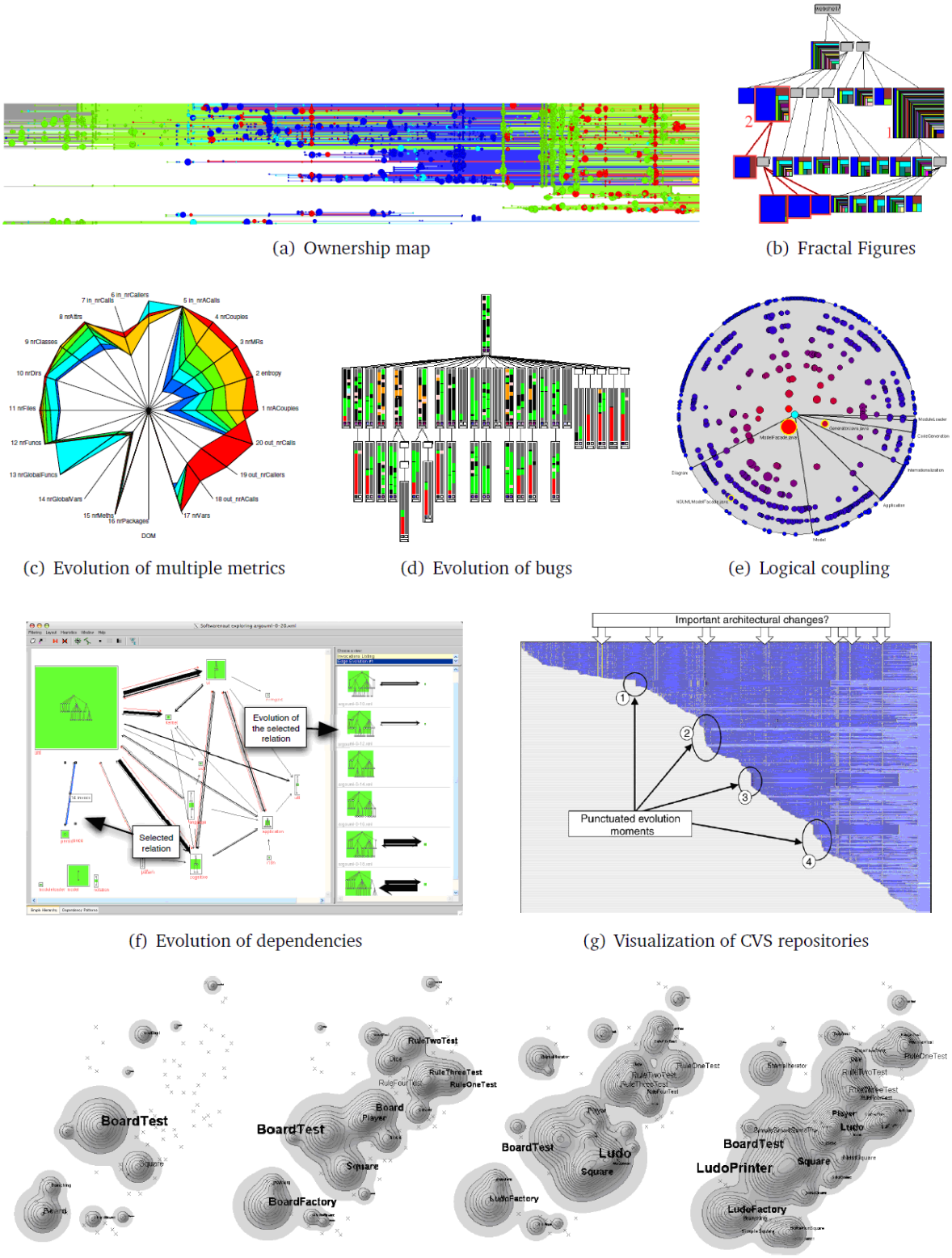
ソフトウェア進化の視覚化

21 世紀初め以前には、ソフトウェアシステムの進化に関する研究は魅力的なものであったが、データが欠落していたために現実的な見通しが得られていなかった。しかし、オープンソースへの動きが広がるとともに、ソフトウェアシステムの全開発記録を含むバージョン管理用リポジトリの数は増え続け、公けにアクセスできるようになった。そ

の結果、多くの研究者が仕事の焦点をソフトウェア進化に向けるようになった。

ソフトウェア進化におけるこの新しい方向を示す以下のような貢献がある。

●2001 年に Lanza はクラスの進化を分析するために Evolution Matrix の polymetric view を使ってクラスの進化のパターンを明らかにした。
●2002 年に Taylor と Munro によって提案された Revision Towers は、バージョン管理リポジトリのログファイルから抜き出したファイルレベルでの変更情報を示すものであった。
●Wu 他による Evolution Spectrographs は単一の属性に着目してシステムの構成要素の進化を示した。
●Fischer と Gall 他は大規模なソフトウェアの機能の進化を視覚化した。
●Pinzger 他は RelVis の中にカプセル化した複数の進化メトリクス（multiple evolution metrics）を実装した（Fig 2.11(c)）。
●Ratzinger 他は EvoLens を表した。時間を含めることによって作られる巨大な探査空間の中のナビゲーションを容易にすることを目的とした技術である。
●Gîrba 他は“Class Hierarchy History Complexity polymetric view”を使ってクラス階層の進化について研究した。
●開発者に焦点を当てた 2 つの興味あるアプローチがある。一つは Gîrba 他の Ownership Maps と呼ばれる視覚化を使ったコード主権（ownership）の進化に関する研究である。もう一つは D'Ambros 他による Fractal Figures であり開発成果（effort）に焦点を当てている。
●D'Ambros と Lanza は BugsCrawler によってソフトウェアバグの変化を視覚化した（Fig 2.11(d)）。また、Evolution Radar を用いてクラス間の論理的な組み合わせ（同時に変化するクラス）を視覚化した（Fig 2.11(e)）。
●Lungu と Lanza は Softwarenaut を用いてモジュール間関係の進化を視覚化した（Fig 2.11(f)）。
●Telea と Auber は 1 行単位のコードのレベルまで細かい粒度でソースコードの変化を追跡することができる Code Flows と呼ばれる技法を提案した。
●Voinea 他は専用のツール CVSscan と CVSgrab を使って CVS リポジトリから抜き出したシステム進化の概観を示した（Fig 2.11(g)）。
●地図製作法に刺激されて、Kuhn 他は Software Cartographer と呼ばれるツールを使ってアプリケーションの語彙の進化を視覚化した（Fig 2.11(h)）。



(a) Ownership map　(b) Fractal Figures

(c) Evolution of multiple metrics　(d) Evolution of bugs　(e) Logical coupling

(f) Evolution of dependencies　(g) Visualization of CVS repositories

(h) Evolving software map

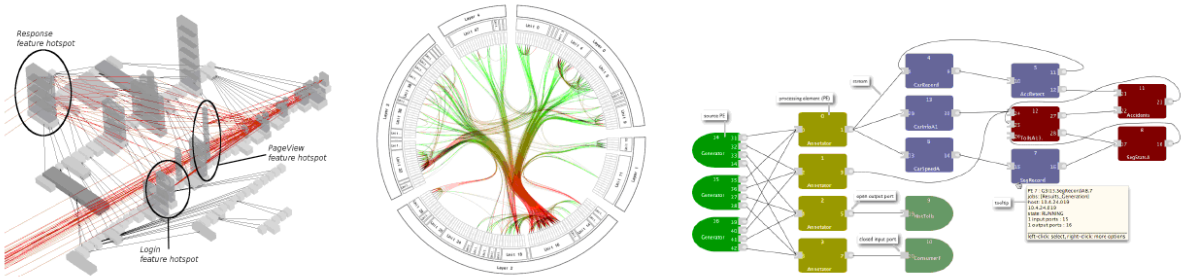
<<Figure 2.11. visualization of software evolution>>

進化に対して向けられた豊かな、多くの視覚化に着目して、Diehl はソフトウェア視覚化のアプローチを 3 つのカテゴリーに分割した。従来 Price 他によって確立されていたプログラムの静的および動的な視覚化の方向に加えて、ソフトウェアシステムの進化の視覚化を含めたのである。

動的なソフトウェア視覚化

動的なデータを視覚化するときには膨大な量のデータを取り扱う必要があり、スケーラビリティが最大の課題である。今日では利用できる計算能力が増大し続けており動的な視覚化が現実的なものとなり始めている。

Greevy 他は装備（feature）間の相互作用を分析するための TraceCrawler（Fig 2.12(a)）と呼ばれる 3D の動的視覚化ツールを提案した。Holten 他の ExTraVis は、非常に興味を引く視覚化を提供するもう一つのトレース分析ツールである（Fig 2.12(b)）。De Pauw 他は大規模なストリーミングアプリケーションを視覚化することを目的として Streamsight と呼ばれるツールを発表した。



(a) TraceCrawler　(b) ExtraVis　(c) Streamsight

<<Figure 2.12. Modern dynamic visualization >>

Web 上のソフトウェア視覚化

この 10 年間で多くの国でインターネットが基本的な道具（commodity）となった。ある研究者達は、彼らのソフトウェア視覚化を web アプリケーションの形で web 上に移した。目的は、彼らの技術へのアクセシビリティを高め、従来のツールの導入や設定に関連する固有の問題を緩和することである。

Mesnage と Lanza はバージョン管理システムのリポジトリに蓄えられたデータに基づいてソフトウェア進化視覚化のための White Coats という 3D web アプリケーションを開発した（Fig 2.13(a)）。

Anslow 他は 3D ソフトウェア視覚化とアニメーションの媒体として web を追究した。

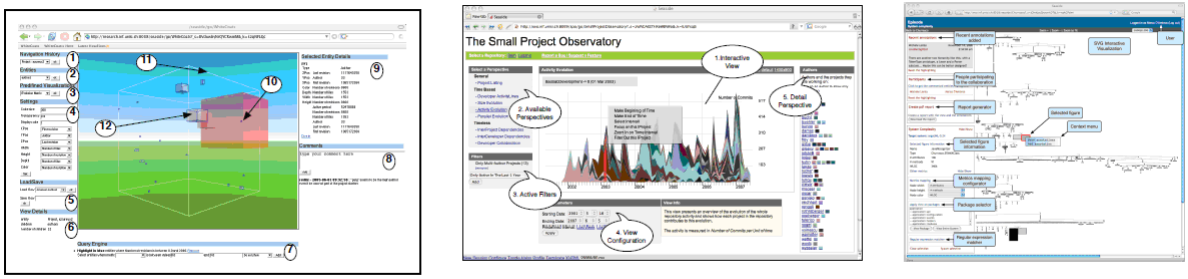
Lungu 他は Small Project Observatory（Fig 2.13(b)）を表した。全体として焦点を当

てているのは、生態系としてのソフトウェアのリバースエンジニアリングを支援することである。

D'Ambros と Lanza は Churrasco（Fig 2.13(c)）を実装した。ソフトウェア視覚化と、並列した注釈（annotation）とを結びつける方法を使って、進化するソフトウェアシステムの動きを、協働して、視覚的に分析することを支援するツールである。

配置可能な（configurable）ソフトウェア視覚化

ソフトウェア視覚化が広く利用されるようになり、豊富なデータが利用可能になったことにより多くの研究者が高度に配置可能な視覚化エンジンを構築することに努力を払った。新しい視覚化のプロトタイピングを手早く行ったり、既存の視覚化を新しい型のデータに適用することができるようにするためである。そのようなエンジンの 2 つの例がある。Favre による GSEE と、Meyer 他によるソフトウェア視覚化をインタラクティブに構築することのできる視覚化フレームワーク Mondrian である。



(a) White Coats　(b) The Small Project Repository　(c) Churrasco

<< Figure 2.13. Software visualizations as web applications >>

ソフトウェア視覚化の統合（integration）

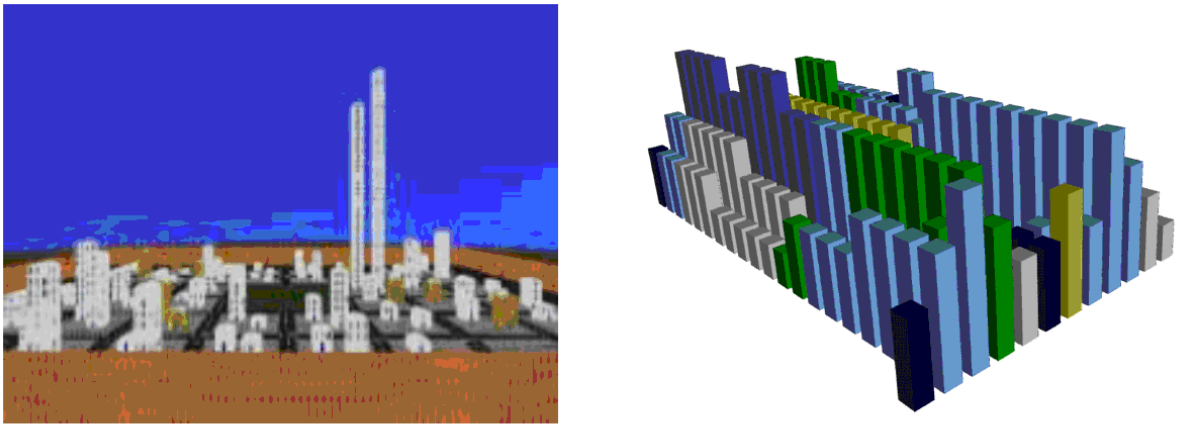
広く使われている IDE（統合開発環境）に統合することによって多くの研究者が彼らの視覚化を実務者により近いものとしていった。Eclipse におけるそのような統合化の例の中に、Lintern 他の SHriMP views の Eclipse プラグインである Creole や、Eclipse において polymetric views を統合するためのプラグインである Malnati の X-Ray がある。

3D メタファーに基づく視覚化

この 10 年間のハードウェアの進歩によって、ソフトウェア視覚化の 3 次元ソリューションが大幅に増加した。初期の調査報告に記述されている“第一世代”の 3D ソフトウェア視覚化アプローチの例は広く使われている 2D グラフベース表現の拡張がほとんどであって、本来の 3D の能力の効果をもたらすものではなかった。しかし、現在最先端のアプローチは 2D 視覚化に刺激されたメタファーをはるかに超えて、次に述べるように深い経験によりよく適合する新しいメタファーを提示している。

2000 年に Knight 他は Software World と呼ばれる VR のための 3D メタファーを発表し、実現した。これによるとシステムは世界として表現され、ソースファイルは都市（city）、クラスは地区（district）、メソッドはビルディングとして表現される（Fig 2.14(a)）。

2001 年に Maletic 他は VR 環境におけるオブジェクト指向ソフトウェアを視覚化するシステムである Imsovision を提示した。その後この経験をもとに Marcus 他は、立体棒グラフ（poly cylinders）の概念を中核としてソフトウェア視覚化の新しい 3D 表現を提案し、sv3D の中に実装した（Fig 2.14(b)）。

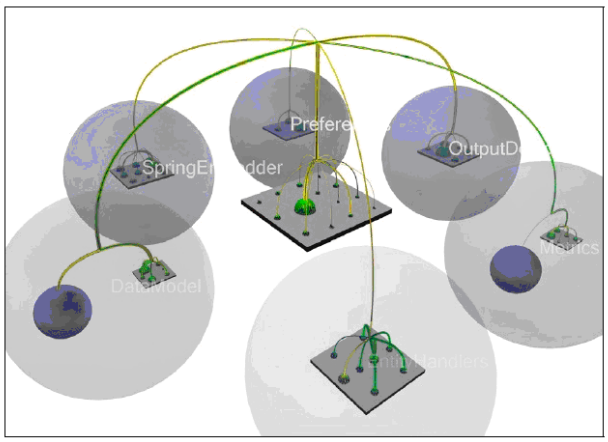


(a) Software World: a city representing a source file　　(b) sv3D: a poly cylinder representing a source file

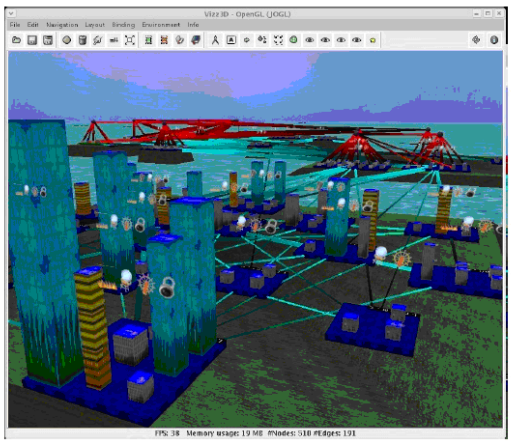
<<Figure 2.14. Software visualization based on 3D metaphors (1/3)>>

2004 年に Balzer は風景（landscape）メタファーに基づいて Software Landscapes 視覚化を提案した（Fig 2.15(a)）。風景という親密なメタファーによって直観的なナビゲーションや包括性を容易にしようとするものであった。

2003 年に Panas 他はソフトウェアの静的、動的両方の側面を描く 3D の city メタファーを考えドラフトを書いた。2 年後彼らは、様々な 3D メタファーや、プロトタイプを実装することのできる Vizz3D と呼ばれる枠組み（フレームワーク）のアーキテクチャによってこのアイデアをフォローアップした。さらに 2 年後、Panas 他はシステムの全体像（overview）の重要性に気付き、彼らの city メタファーに基づいて unified single-view 視覚化を提案した。



(a) Software Landscapes: a landscape metaphor

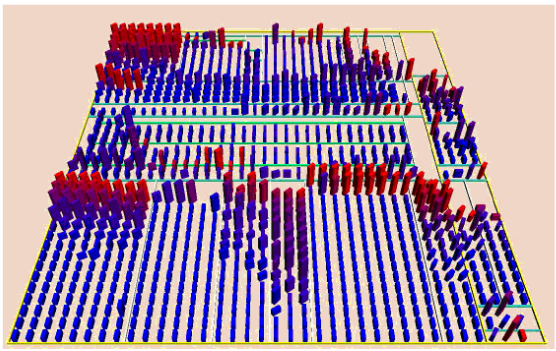


(b) Vizz3D: a city metaphor

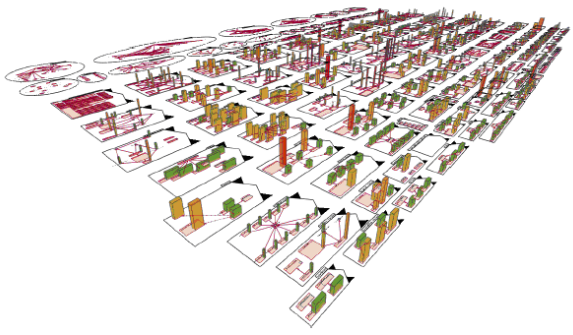
<<Figure 2.15. Software visualizations based on 3D metaphors (2/3)>>

風景的なメタファーを使った 3 つ目のアプローチは Langelier 他のものである（Fig 2.16(a)）。Verso と呼ばれる実装では、測度を視覚的な属性として写像した 3D の箱としてクラスを表現し、ソフトウェアシステムの構造を表現するために 2 つの空間に効率的に配置した（two space efficient layouts）。

最終的に、Lange と Chaudron はソフトウェア測度で UML ダイアグラムを強化し一連の視覚化（view）を得た。その中には UML-city と呼ばれる city メタファーに基づく 3D の視覚化も含まれている（Fig 2.16(b)）。



(a) Verso: 3D boxes



(b) UML City: UML diagram enriched with metrics

<< Figure 2.16. Software visualizations based on 3D metaphors(3/3)>>

2.5.1　Evidence　Quest（根拠・証拠・しるし・形跡　の　追求）

この 10 年、ソフトウェアエンジニアリングのアプローチが効率的、効果的であるということの根拠を追求することに対する関心が高まりつつある。Wohlin 他はソフトウェアエンジニアリングにおける実験という課題に取り組んだ。Kitchenham 他はソフトウェアエンジニアリングにおける経験的研究に対する一連の予備的なガイドラインを起草した。2 年後 Kitchenham 他は、医学に適用して成功したことをきっかけとして、ソフトウェアエンジニアリングに“証拠に基づいた（evidence-based）”方法論を採用すべきであると提案した。

ソフトウェアエンジニアリングにおいて、経験的な実証が必要であるという要求が増大し続けているという現実は、否応なくソフトウェア視覚化分野にも及んでいる。副作用の一つとして、ソフトウェアエンジニアリングの主要な流れから視覚化アプローチを排除するときに、経験的な実証が無いということが（しばしば盲目的に）レビュア達によって使われる最終的な理由となっている。逆に、単に統制された実験（controlled experiments）を含んでいるということだけで（特に“統計的に目立ったところ”がある場合には）、ソフトウェア視覚化の論文を“respectable（かなりよい）”にする魔法のような効果がある。

他のソフトウェアエンジニアリング分野と比較して、ソフトウェア視覚化を経験的に実証することがそれほど難しいのはなぜなのか？

多くのソフトウェアエンジニアリング分野は明確に定義された問題と仕事の枠組み（すなわち、容易にブラックボックスとして抽象化することができる）があり、オペレーションは人手の介入を必要とせず（すなわち自動化されている）、成果は決定論的である。

対照的に、ソフトウェア視覚化アプローチは典型的に（対象とする活動が拡散するという性質の故に）明確に定義された問題や仕事の枠組みに欠けており、オペレーション中に人手の介入を必要とし（すなわち、半自動化である）、成果は観察力や視覚など解析的な人間の技量に依存する。ソフトウェア視覚化だけで問題が解決することはまれである。そうではなく、ソフトウェア視覚化は人間が問題を解決するプロセスを支援するものである。ソフトウェア視覚化の効率性、効果性等の貢献度を測るということは極めて難しい課題である（a challenge）。

Maletic 他が指摘したように、この難しさの徴候はソフトウェア視覚化のベンチマーク（水準点）が全く無いということにみられる。いくつものソフトウェアエンジニアリ

ングの分野では研究者が新しいアプローチの成果を最先端の技術と比較できるような専用のベンチマークがある（例えば、分散システム）。一方、ソフトウェア視覚化にはベンチマークが無い。ベンチマークが無いということの結果、様々な実験を設定するために必要となる労力は増大し、標準化が行われていないために実証することに対する圧力の数も増大している。

不利な状況にもかかわらず、この 3 年間でいくつものソフトウェア視覚化アプローチの経験的な実証が注目されるようになってきた。一つの経験的な実証が成功すれば、その成功はあらゆるソフトウェア視覚化アプローチに対して有利に働く。その意識に基づいて、実験の権威者たちのほとんどは彼ら自身の視覚化アプローチを実証することを目指した。Lange 他による UML モデルにおけるインタラクティブな視覚化、Quante による dynamic object process graphs、Cornelissen による trace 視覚化などである。

研究者たちは、他の研究者によって開発されたツールを評価することによって視覚化の一般的な識見（insight）を追求した。例えば、Sensalire 他は、視覚化ツールの望ましい装備（feature）のモデルを構築することをゴールとして、いくつかの既存のツールを使うことを含めた一連の評価のための実験を行った。

疑いも無く、経験による実証はソフトウェア視覚化の領域にまでおよんでいる。経験による実証がこの領域の将来に対して長期的にどのような影響を与えるかを考えることは意義深いことであろう。

2.6 Summary（まとめ）

この章では、我々はソフトウェア視覚化の歴史的な側面について述べた。それらは全て過去 50 年前に始まっており、図（ダイアグラム diagram）、pretty-print や動画を含めてプログラムの行動を描いた。1980 年代には進化した pretty-print に焦点が移ったが、同時にアルゴリズムを用いたアニメーションやいくつかの初期の静的な視覚化アプローチなどがあった。1990 年代には視覚化されるべきソフトウェアシステムの局面に合わせてソフトウェア視覚化研究の興味の重点は 3 つの異なった方向に整理されていった：①静的な情報、②動的な情報、③システム進化（evolution）である。しかしながらソフトウェア視覚化の隆盛に向けての転機は 2002 年であった。この分野における主要な研究者たちのほとんどがソフトウェア視覚化に特化した一連の立場を立ち上げることに専念したのである。

これらはソフトウェア視覚化の研究にとってエキサイティングな（刺激的な、興奮させる、面白くて仕方がない）時期であった。一方ではハードウェアにおける進歩が、20 年前にはわずかに想像されていたにすぎない新しい地平線を開いた。他方、ソフトウェア視覚化アプローチ（その

うちのあるものは役に立たないものであったが）が浸透し始め、それに対する懐疑主義を生み、その利用可能性を評価する仕組み（method）を求め始めた。

我々の論文の最初の主張は、ソフトウェアシステムを都市（city）として書き表すことが、ソフトウェア視覚化に対して非常に広範囲に適用可能なメタファーであるということである。PartII では city メタファーに基づく我々のアプローチと、我々がメタファーの適用可能性を明示する（demonstrate）ために使う 3 つのアプリケーションについて説明する。

Ⅲ 活動報告

(1) 例会

2012 年 8 月 17 日金曜日、名古屋市東区の中電シーティーアイ会議室で B グループミーティングを開催しました. タイムテーブルは以下の通りです.

17:00～18:45　SERC-B グループ ミーティング
18:45～19:00　移動
19:00～21:00　暑気払い　場所：未定

このミーティングには、４名（石川、諸岡、江尻、木部）が参加し、内容は,

- 昨年の SERC シンポジウムでの B グループ発表の振り返り
- 参加者各人の現在の、問題意識の意見交換
- B グループ活動報告書に関する議論
- 研究成果の中間報告

を行ないました。
ミーティングの後は場所を改めて、交流を深めるための暑気払い会を行ないました。

Ⅳ 今後の予定

今後の活動については、以下を基本的な方針とします。

- ソフトウェア・ツールの研究
  - ✧ ソフトウェア保守・進化に有用なツールを発掘し、評価する
- ソフトウェア・サステナビリティの研究
  - ✧ ソフトウェア・サステナビリティの動向調査
  - ✧ ソフトウェア・サステナビリティの概念を把握する活動
- フォーラムの開催
  - ✧ 参加者と研究員とのディスカッション要素をより多く取り入れたフォーラムの開催
- その他、ソフトウェア進化・保守を支える考え方の調査
  - ✧ ソフトウェア保守開発以外の分野に関して、ソフトウェア保守・進化と共通の性格を持つ、考え方や技法の調査、把握

これらの研究を行い、フォーラム開催やインターネット上への(中間)成果公開をより頻度高く推進していくことを基本的な方針としたいと思います.

第21年度ＳＥＲＣ報告書

グループ名：

ＳＥＲＣ－Ｃグループ

研究テーマ：

「１０年後のソフトウェア保守を考える」

参加メンバー：

㈱アイ・ティ・フロンティア	田中　創
㈱アイ・ティ・フロンティア	玄間　稔
㈱アイ・ティ・フロンティア	丸山　陽一
中央コンピューター㈱	伊藤　順一
アイエックス・ナレッジ㈱	岡田　浩
アイエックス・ナレッジ㈱	井瀬　英晶
東芝ソリューション㈱	佐井　由美子

２０１２年9月２１日

目次

１．はじめに
　１－１．活動の経緯
　１－２．フォーラム開催報告
２．１０年後を考えた保守要員の人材育成
　２－１．ノウハウ伝承と人材育成の課題
　２－２．ＩＴＳＳにおける保守ＳＥのスキル標準
　２－３．保守現場のスキル標準
　２－４．保守ＳＥの育成
３．１０年後のソフトウェア保守を考える「ソフトウェア保守の自動化を目指して」
　３－１．未来型自動車：自走運転システム
　３－２．ＩＴ業界の運用の自動化動向
　３－３．ＳＩビジネスの１０年後とソフトウェア保守は？
４．「１０年後の保守環境(デバイスフリーがもたらす保守環境への影響について)
　４－１．デバイスフリー端末の利用
　４－２．保守の現場における現状の環境
　４－３．現状の保守現場の問題点
　４－４．問題点を解決するためのデバイスフリー端末の活用
　４－５．まとめ
５．私が考える１０年後のソフトウェア保守
　５－１．はじめに
　５－２．過去から現在
　５－３．１０年後の予測
　５－４．１０年後のソフトウェア保守技術者
　５－５．若手エンジニアへの伝承の必要性
　５－６．若手エンジニアへ伝承すべきものごと
　５－７．伝承のためのステップ
　５－８．考察を終えて
６．保守開発のグローバル化：１０年後の国内ＳＩビジネスと保守開発者の心構え
　６－１．はじめに
　６－２．１０年後の事業環境
　６－３．１０年後のシステム環境
　６－４．１０年後の保守開発グローバル化
　６－５．１０年後への備え
　６－６．１０年後の保守開発者への提言

７．１０年後の保守担当に求められるスキル

７－１．１０年前のＳＥ

７－２．現在のＳＥ

７－３．１０年前から現在に至って

７－４．１０年後（２０２２年）の保守

７－５．１０年後（２０２２年）の予想

８．終わりに

８－１．今年度の総括

８－２．次年度に向けて

【参考文献】

・朝日新聞社「朝日新聞」2011.7.15

・産経新聞社「産経ニュース」2012.5.9

・独立行政法人　情報処理推進機構　ＨＰ　　http://www.ipa.go.jp/

・海外 IT アウトソーシングの進め方とポイント（案）　：　JISA 国際委員会日中部会

・ＳＩと運用が消える　：　日経コンピュータ 2012.6.7 号

・激動 トヨタ ピラミッド　：　ＮＨＫスペシャル

１．はじめに

１－１．活動の経緯

Ｃグループは、2003 年より「保守のアウトソーシングを考える」をメインテーマとして保守現場に密着したサブテーマを毎年選択し研究活動を継続してきた。直近５年間は以下のテーマについて議論してきた。

2007 年「保守スキル標準化による要員ローテーションの取り組み」
2008 年「内部統制と要員ローテーションの活性化」
2009 年「ＳＬＡを支えるＳＥ像」
2010 年「保守のサービスレベル向上策」
2011 年「障害を発生させない、被害を拡大させない対策とは」

「保守のアウトソーシングを考える」をメインテーマに据えて９年が経過する中で、経済環境も大きく変化した。そこで、次の１０年へ向けてメインテーマを新たに設定し、2012 年からは、「１０年後の保守を考える」をメインテーマに、毎回フォーラム形式を主体とした活動とし、討論の中から導き出されたサブテーマをさらに議論していきたいと考えた。

１－２．フォーラム開催報告

Ｃグループでは、今年度「１０年後のソフトウェア保守を考える」をテーマとして、ゲストスピーチの講演をメインにフォーラムを２回開催した。
今年度の報告書は、フォーラム報告とＣグループ員がいろいろな観点から見た１０年後のソフトウェア保守のテーマを纏め報告書とした。

【第１回フォーラム】
１．開催日時：２０１２年３月８日（木）
２．開催場所：アイ・ティ・フロンティア　会議室
３．講演
①ゲストスピーチ
「１０年後の保守を考える」　高橋芳宏氏（ＳＥＡ幹事）
＊内容は今年度の研究会報告書に掲載のフォーラム資料を参照
②Ｃグループからの提言
「１０年後の保守を考える」　伊藤順一氏（Ｃグループ）
＊内容は今年度の研究会報告書に掲載のフォーラム資料を参照

４．参加者からの意見

①今回の震災の教訓、保守の在宅勤務は可能か？

・時間管理など労務管理が大変

・保守業務は仕事の内容からコミュニケーションを取りながら行うので難しい

②ビジネスロジックの伝承は必要か？

・仕様書があてにならないので、ソースコードを読むように指示

・ソースコードを読む時に業務知識が必要で重要である

・顧客から業務知識を引き出すのに大変苦労する

③保守でツール（調査用やソースコードジェネレータ）の活用は？

・ワークロードかかる調査分析やテストでツールの活用が出来れば生産性は上がる

・１９９０年代当初、リバースツールやＣＡＳＥツールが脚光を浴びたが、手を加えると自動生成が出来なかった。現在もツールで作成した本体の改修は手作業では行なわない、業務をパッケージに合わせる、外付けで開発するなど、オフショアも意識した考慮が必要

④保守ビジネスの発展性

・保守で大切なことは、医者と同じように顧客の悩みを訊き、ＢＰＲを実施し提案をしていく関係を堅持していくことが重要

・基幹系ではないパッケージソフトなどの保守は、クラウド型も含めたアウトソーシング型に移行してくのではないか

【第２回フォーラム】

１．開催日時：２０１２年7月２５日（水）

２．開催場所：東京都港区芝浦港南区民センター　会議室

３．講演

①ゲストスピーチ

「ウラから観た、ソフトウェア保守の過去・未来・現在」

田中一夫氏（ＳＥＲＣ事務局）

＊内容は今年度の研究会報告書に掲載のフォーラム資料を参照

②Ｃグループからの提言

Ｃグループ各参加者によるスピーチ

＊内容は今年度の研究会報告書に掲載のフォーラム資料を参照

４．参加者からの意見

①２０年間を振り返り保守はあまり変わっていない

・保守の定義が曖昧な時期もあり、きちんとしたプロセスが定義出来ていなかった

・保守教育は重要であるが、テキストもなく、各社各個人固有の方法で教育を行なっているのが現状で、保守教育の出版物も殆ど出ていない

②１０年後のソフトウェア保守の展望

・戦略的システムの保守は現行の延長で残るだろう

・戦略的システム以外のパッケージソフトウェアなどは、今後クラウドを活用し、顧客の運用・保守は無くなっていくと思われる

２．１０年後を考えた保守要員の人材育成

２－１．ノウハウ伝承と人材育成の課題

2007 年 8 月 31 日付のＣグループ報告書「保守スキル標準化による要員ローテーションの取り組み」にて、保守の現場においても経済産業省のＩＴスキル標準（ITSS）と関係付けて保守スキル標準を設定し、人材育成計画を策定するとともにキャリアパスとローテーションを考える必要があると提言した。

また、産業能率大学が実施した意識調査から「自分の技能や知識を社内で伝承できていない」と感じている会社員が多く、「シニア層のノウハウを伝承する仕組みを意図的に作ることが不可欠」と有識者から指摘されている。

従って、保守の現場でも保守のノウハウが伝承されていないことが想像できる。これからの保守を考えた時、無人化の問題が保守の現場の生産性や品質に大きな影響を与えるのではと杞憂される。

一時、2007 年問題も話題に上って、日本企業で基幹系システム（大型汎用機やオフコンによるレガシーシステム）を最初に構築し、これまで運用・保守を行ってきたベテラン・エンジニアがそろって定年を迎え、今後の企業システムのメンテナンスが困難になると騒がれた。

ＩＴの世界では、システム開発の主流は汎用機からオープン系へ移行しているため、若手エンジニアで汎用機や COBOL の知識を持っている人は多くない。システム開発運用の現場では、ベテラン・エンジニアがレガシーシステムを担当し、若手がクライアント／サーバーシステムや Web システムを取り扱うケースが多いため、技術的にも業務知識的にもノウハウの継承ができていないことがほとんどだ。

さらに、最近のクラウドサービスの拡大でコスト削減の要求は強くなり、システム基盤の多様化とシステム構造の複雑化による保守要員の人材不足が大きな問題となるであろう。

本章では、これから１０年後の保守を考えた時、誰が誰に何を伝承すべきなのか、どのような人材育成を目指すべきなのか考えてみたい。

２－２．ＩＴＳＳにおける保守ＳＥのスキル標準

ＩＴスキル標準（ITSS）は、情報サービス業界で働く人材に求められる「スキル」、すなわち知識や経験、実務能力を体系的に整理したもので、目的は個人の自律的なスキル向上や企業における人材育成を促し、結果として日本のＩＴサービス産業の競争力向上を実現することにある。2006 年 4 月に初版、2006 年 10 月にＶ２、2008 年 3 月にＶ３が、2011 年 3 月にはＩＴスキル標準Ｖ３　2011 が公表されている。

その代表的な資料である図２－１「ＩＴスキル標準Ｖ３　キャリアフレームワーク」で

は、職種と専門分野が定義されている。
図２－２「職種とタスクの関連図」と表２－３「職種の説明一覧」から見えるように業務システム系保守に関しては「ＩＴスペシャリスト」と「アプリケーションスペシャリスト」の活動領域に定義されている。

では、保守ＳＥに必要なスキルとは何かを求めるため、ＩＴスキル標準Ｖ３の「スキル領域とスキル熟達度」のドキュメント上の「ＩＴスペシャリスト」と「アプリケーションスペシャリスト」のスキル領域（スキル項目と知識項目）に目をやると、残念ながら保守に関する定義はない。

それでは、保守ＳＥに必要なスキルとは何か、以降で考察していきたい。

図２－１　ＩＴスキル標準Ｖ３　キャリアフレームワーク

職種	マーケティング			セールス			コンサルタント		ITアーキテクト			プロジェクトマネジメント				ITスペシャリスト						アプリケーションスペシャリスト		ソフトウェアデベロップメント			カスタマサービス				ITサービスマネジメント				エデュケーション	
専門分野	マーケティングマネジメント	販売チャネル戦略	マーケットコミュニケーション	訪問型コンサルティングセールス	訪問型製品セールス	メディア利用型セールス	インダストリ	ビジネスファンクション	アプリケーションアーキテクチャ	インテグレーションアーキテクチャ	インフラストラクチャアーキテクチャ	システム開発	ITアウトソーシング	ネットワークサービス	ソフトウェア製品開発	プラットフォーム	ネットワーク	データベース	アプリケーション共通基盤	システム管理	セキュリティ	業務システム	業務パッケージ	基本ソフト	ミドルソフト	応用ソフト	ハードウェア	ソフトウェア	ファシリティマネジメント	運用管理	システム管理	オペレーション	サービスデスク	研修企画	インストラクション
レベル7																																			
レベル6																																			
レベル5																																			
レベル4																																			
レベル3																																			
レベル2																																			
レベル1																																			

出典：ＩＴスキル標準Ｖ３ 2011　２部:キャリア編（ＩＰＡ）

図２－２　職種とタスクの関連図

IT投資の局面と活動領域 ＼ 職種	経営戦略策定		戦略的情報化企画		開発		運用・保守	
	経営目標／ビジョン策定	ビジネス戦略策定	課題整理／分析（ビジネス/IT）	ソリューション設計（構造/ﾊﾟﾀｰﾝ）	コンポネント設計（ｼｽﾃﾑ/業務）	ソリューション構築（開発/構築）	ソリューション運用（ｼｽﾃﾑ/業務）	ソリューション保守（ｼｽﾃﾑ/業務）
セールス	目標/ビジョンの確認	ビジネス戦略の確認	ビジネス課題ｿﾘｭｰｼｮﾝ提案					
コンサルタント	目標/ビジョンの提言	ビジネス戦略策定の助言	ソリューション策定のための助言	ソリューションの設計				
IT アーキテクト			ソリューションの枠組み策定	ソリューションｱｰｷﾃｸﾁｬｰの設計	コンポネントの設計	ソリューションの構築		
プロジェクトマネジメント			プロジェクト基本計画の策定	プロジェクトの管理/統制	プロジェクトの管理/統制	プロジェクトの管理/統制	プロジェクトの管理/統制	プロジェクトの管理/統制
IT スペシャリスト				システム構築計画の策定	システム・コンポネントの設計	システム・コンポネントの導入構築	システム・コンポネントの運用支援	システム・コンポネントの保守
アプリケーションスペシャリスト				ｱﾌﾟﾘｹｰｼｮﾝ開発計画の策定	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの設計	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの開発	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの運用支援	ｱﾌﾟﾘｹｰｼｮﾝコンポネントの保守
カスタマサービス					導入計画の策定	ハードウェアソフトウェアの導入	ハードウェアソフトウェアの保守	ハードウェアソフトウェアの保守
ITサービスマネジメント						運用計画/運用管理の策定	システムの運用と管理	システムの運用と管理

主たる活動局面　　従たる活動局面

出典：ＩＴスキル標準Ｖ３ 2011　２部:キャリア編（ＩＰＡ）

ＩＴスキル標準で定義している各職種の説明を次表に示す。各職種が社会的なプロフェッショナルとして確立していくことを狙う観点から、当該職種に求められる成果と品質について定義している。

表２－３　職種の説明一覧

職種	概　要
マーケティング	顧客ニーズに対応するために、企業、事業、製品及びサービスの市場の動向を予測かつ分析し、事業戦略、販売戦略、実行計画、資金計画及び販売チャネル戦略等ビジネス戦略の企画及び立案を実施する。市場分析等をつうじて立案したビジネス戦略の投資効果、新規性、顧客満足度に責任を持つ。
セールス	顧客における経営方針を確認し、その実現のための課題解決策の提案、ビジネスプロセス改善支援及びソリューション、製品、サービスの提案を実施し成約する。 顧客との良好なリレーションを確立し顧客満足度を高める。
コンサルタント	知的資産、コンサルティングメソドロジを活用し、顧客の経営戦略やビジネス戦略及びＩＴ戦略策定へのカウンセリング、提言、助言の実施を通じて、顧客のビジネス戦略やビジョンの実現、課題解決に貢献し、ＩＴ投資の経営判断を支援する。 提言がもたらす価値や効果、顧客満足度、実現可能性等に責任を持つ。
ＩＴアーキテクト	ビジネス及びＩＴ上の課題を分析し、ソリューションを構成する情報システム化要件として再構成する。ハードウェア、ソフトウェア関連技術（アプリケーション関連技術、メソドロジ）を活用し、顧客のビジネス戦略を実現するために情報システム全体の品質（整合性、一貫性等）を保ったＩＴアーキテクチャを設計する。設計したアーキテクチャが課題に対するソリューションを構成することを確認するとともに、後続の開発、導入が可能であることを確認する。また、ソリューションを構成するために情報システムが満たすべき基準を明らかにする。さらに実現性に対する技術リスクについて事前に影響を評価する。
プロジェクトマネジメント	プロジェクトマネジメント関連技術、ビジネスマネジメント技術を活用し、プロジェクトの提案、立上げ、計画、実行、監視コントロール、終結を実施し、計画された納入物、サービスと、その要求品質、コスト、納期に責任を持つ。

ＩＴスペシャリスト	ハードウェア、ソフトウェア関連の専門技術を活用し、顧客の環境に最適なシステム基盤の設計、構築、導入を実施する。 構築したシステム基盤の非機能要件（性能、回復性、可用性など）に責任を持つ。
アプリケーションスペシャリスト	業種固有業務や汎用業務において、アプリケーション開発やパッケージ導入に関する専門技術を活用し、業務上の課題解決に係わるアプリケーションの設計、開発、構築、導入、テスト及び保守を実施する。構築したアプリケーションの品質（機能性、回復性、利便性等）に責任を持つ。
ソフトウェアデベロップメント	ソフトウェアエンジニアリング技術を活用し、マーケティング戦略に基づく、市場に受け入れられるソフトウェア製品の企画、仕様決定、設計、開発を実施する。また上位レベルにおいては、ソフトウェア製品に関連したビジネス戦略の立案やコンサルテーションを実施する。開発したソフトウェア製品の機能性、信頼性等に責任を持つ。
カスタマサービス	ハードウェア、ソフトウェアに関連する専門技術を活用し、顧客の環境に最適なシステム基盤に合致したハードウェア、ソフトウェアの導入、カスタマイズ、保守（遠隔保守含む）、修理を実施するとともに、顧客のシステム基盤管理およびサポートを実施する。またＩＴ施設インフラの設計、構築、導入および管理、運営を実施する。導入したハードウェア、ソフトウェアの品質（使用性、保守容易性等）に責任を持つ。
ＩＴサービスマネジメント	システム運用関連技術を活用し、サービスレベルの設計を行い顧客と合意されたサービスレベルアグリーメント（SLA）に基づき、システム運用リスク管理の側面からシステム全体の安定稼動に責任を持つ。システム全体の安定稼動を目指し、安全性、信頼性、効率性を追及する。またサービスレベルの維持、向上を図るためにシステム稼動情報の収集と分析を実施し、システム基盤管理も含めた運用管理を行う。
エデュケーション	担当分野の専門技術と研修に関連する専門技術を活用し、ユーザーのスキル開発要件に合致した研修カリキュラムや研修コースのニーズの分析、設計、開発、運営、評価を実施する。
共通（レベル１、２）	担当業務の技術領域に関する基本知識を活用し、上位者の指示の下、あるいは既存の作業標準やガイダンスに従い、要求された作業を実施する。自らの担当作業に対する実施責任を持つ。

出典：ＩＴスキル標準Ｖ３ 2011　２部:キャリア編（ＩＰＡ）

２－３．保守現場のスキル標準

図２－４「ソフトウェア保守プロセスと工数および時間の相関関係」は、JIS　X0161で規格化された「保守プロセス」にSERCが提唱する「保守の工数と時間の相関関係」を示す、通称「フタコブらくだ」を重ね合わせたものである。この図では、ソフトウェア保守プロセスの中で一番に工数を要するのは「②問題の把握および修正分析」と「④保守レビューおよび受け入れ」であることを表している。通常のソフトウェア開発においては、「③修正の実施（開発プロセスの呼び出し）」が「ヒトコブらくだ」を連想するように、工数の山積で一番高いのは開発プロセスのプログラム製造である。

保守ＳＥに必要なスキルは、図２－４で示すプロセスを遂行できる知識や経験であり、保守対象システムの特性により、そのレベルは異なる。

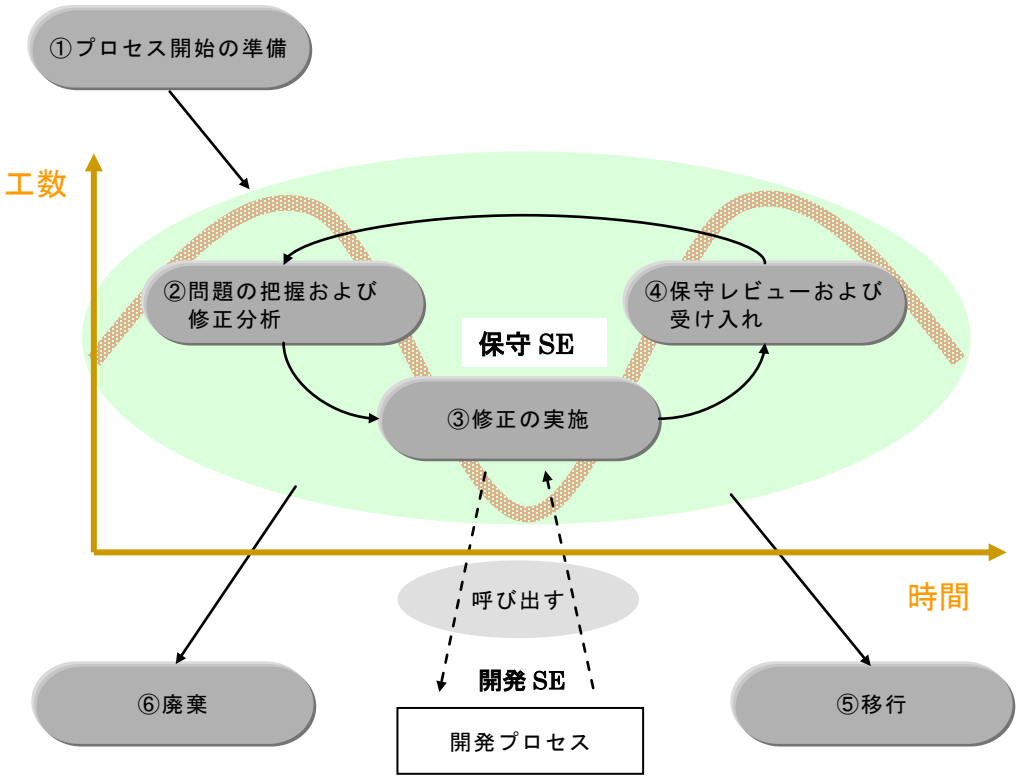
先に述べたＩＴＳＳで定義されているアプリケーションスペシャリストはこのプログラム製造の前工程としての要件定義、基本設計や後工程としての結合テスト、システムテストに携わる人材であり、これらの開発工程のスキルを有することが必要である。

とりわけ保守ＳＥのスキルとしては、これらのソフトウェア開発工程のスキルを有することが前提であるが、更に前述の「フタコブらくだ」が示すように保守業務で重要な位置づけにある「②問題の把握および修正分析」と「④保守レビューおよび受け入れ」を遂行できる能力が求められる。

特に「②問題の把握および修正分析」においては、保守案件（是正保守、適応保守）の対象となる業務について精通していなければ、適切な影響範囲が特定できず、これを怠れば後々不具合発生の要因を残すことになる。また、「④保守レビューおよび受け入れ」において、不具合や改善点など適切な指摘を行なうには、保守プロセス全般の実務経験が必要である。

従って、保守ＳＥのスキルレベルは図２－４で表している各々のプロセスを経験した期間やシステム規模により評価されることが一般的であると考える。

図２－４　ソフトウェア保守プロセスと工数および時間の相関関係



２－４．保守ＳＥの育成

保守ＳＥを育成するには、ミクロ的には先の保守プロセスを網羅的に経験させて、所謂「垂直型ローテーション」（下流工程から上流工程へシフトさせる育成方法）によりスキルレベルを向上させる策が身近で実現が容易である。

しかし、M&Ａによるシステム統廃合やクラウドサービスへの置き換えなどにより、既存システムの廃棄や別のプラットフォームへの移行等、単一のアプリケーション・コンポネントからシステム・コンポネント（複数のアプリケーションシステムや複数のプラット

フォームから構成される複合体）としての保守対応が多くなると考えられる。このようなシステム・コンポネントの多様化や複雑化に対応していくには、マクロ的な視点での育成計画が重要である。単なる保守プロセスのスキルアップだけではなく、高度なスキルを修得させるためにも、図２－５「保守ＳＥのキャリアパスの例」で示すような職種のドメインを超えたキャリアパスを考える必要がある。

保守ＳＥが目指すべき職種は、ＩＴスペシャリストは基より、ＩＴアーキテクトがより適任であると考える。なぜなら、ＩＴアーキテクトはユーザーとシステムエンジニアの橋渡し役であり、保守ＳＥの役割も図２－４で表されているとおり、ユーザー要件を具体化し、開発プロセスを呼び出すなど、まさにユーザーと開発ＳＥの橋渡し役そのものの機能を担っている。

ＩＴアーキテクトを目指し、必要なスキルを日々の業務から修得できるような育成計画を保守の現場で考えていただきたい。

また先に述べたとおり、今後はシステム・コンポネントとしての保守対応が多くなると考えられるため、特にプラットフォームの異なるシステムが増加していくと、データ連結は複雑となり、保守プロセスにおける「②問題の把握および修正分析」は広範囲で高度な分析作業が要求される。

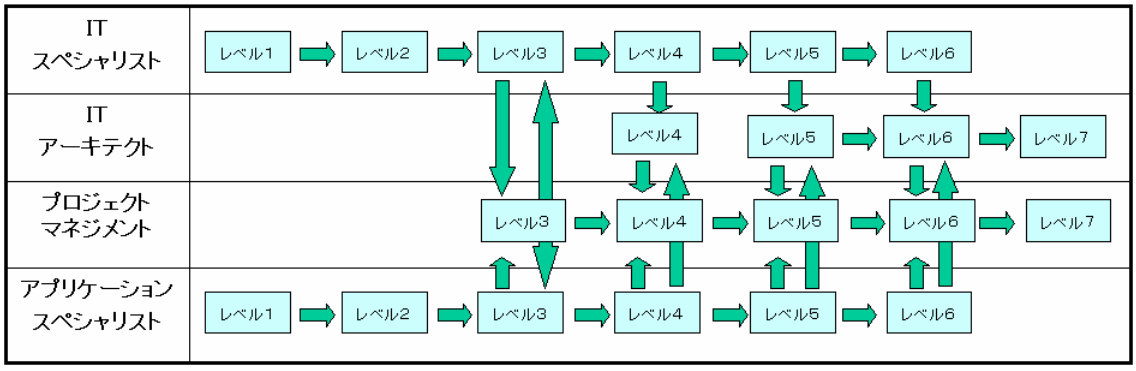
従来の保守ＳＥは単一のアプリケーションシステムを長期間、継続保守することで熟練度を上げ、作業効率と作業品質を向上させてきた。所謂、属人化である。このような保守要員では前述のような高度な保守要求には対応できず、人材不足が大きな問題となるであろう。

その対策としてひとつ考えられるのが、「システム鳥瞰図の整備」である。以前、当Ｃグループにて保守ドキュメントの整備を検討した際、その重要ドキュメントとしてシステム鳥瞰図を採り上げていた。今までは単一プラットフォーム上のアプリケーションシステム関連図（システム鳥瞰図の代表例）で、保守プロセスにおける「②問題の把握および修正分析」に対応できていたが、システム・コンポネントの保守対応ではシステム間のデータ連携の分析が重要であり、データを中心としたより広範囲なシステム関連図が必要となる。

その対応としてベテランが有するスキルや知識を後継者に伝承できるようにこれらのドキュメントの整備に力を注いでいただきたい。そして、これらのドキュメントを利用して保守現場での教育を実施していただきたい。

来るべき１０年後の保守現場でもそれらのドキュメントが有効活用されることを期待している。

図２－５　保守ＳＥのキャリアパスの例



【参考文献】

・ＩＴＳＳ：独立行政法人　情報処理推進機構（ＩＰＡ）ＨＰ　　http://www.ipa.go.jp/

（田中　創）

３．１０年後のソフトウェア保守を考える「ソフトウェア保守の自動化を目指して」

今年度のＣグループのテーマ「１０年後のソフトウェア保守を考える」に関して、少々視点変えて意見を述べさせていただきたいと思う。

今回のテーマにした発端は、最近報道された記事がヒントになり、１０年後までにその一部でも「ソフトウェア保守の自動化」が実現出来ることを願って取り上げてみた。

３－１．未来型自動車：自走運転システム

３－１－１．自走運転システム紹介

「車の運転は、人工知能を活用すれば事故を減らせる」―グーグル社はこのテーマに基づき、人間が直接運転操作をせずに目的地に行く自走運転システムを開発し、２０１１年７月にネバダ州で実際の道路で走行を認める法案を成立させた。そして同州の陸運局は、２０１２年５月８日に２人乗車することを条件に試験運転を許可し、陸運局としては、３～５年後に実用化できるかもしれないと予想しているようである。

この自走車試験では、エンジンと電気モーターを併用し世界で一番売れているハイブリッド車トヨタプリウスを使う。距離を測定しながら屋根や車体の各所にカメラとセンサーを設置し、周りの他の車・歩行者・障害物を避けて交通法規を順守しながら自走運転する。このシステムの開発には、米国防高等研究計画局主催のロボット車競技会に参加した技術者を集め３年以上前から開発してきた。技術のベースになる自走制御はロボット車運行技術で、頭脳部分はグーグル社が得意とする「ストリートビュー」などの地図検索機能と人工知能を使い、車の位置情報と自走時の目となるカメラ、センサーのインプット情報を人工知能で状況判断することにより自動運転が可能となったようである。

この実例でベースになっているのは勿論ＩＴ技術である。目的地に行くためにインプット情報（現在の位置・車の状況・周囲情報）を蓄積された地図情報データベースと照合し、最良なルートを探し出す人工知能を使用して目的地まで自動運転していく。我々が昔見たアニメの世界が一歩ずつ実現して来ているようである。

では、同様にＩＴ技術を使うコンピューター、そのソフトウェア保守の現実はどうだろうか？現在の進化スピードで、１０年後はソフトウェア保守の軽減化・自動化がどれだけ進んでいるのだろうか？

３－１－２．自走運転システムとソフトウェア保守の機能比較

今回実例として取り上げた「自走運転システム」と「ソフトウェア保守」を機能面で比較してみたい。

表３－１　自走運転システムとソフトウェア保守の機能比較

プロセス	自走運転システム	ソフトウェア保守	考察
インプット	①目的地情報 ②位置情報（ＧＰＳ） ③周辺状況（カメラ・センサー） ④車体状況（燃料・スピード・ハンドル操作など）	①障害報告 ②仕様変更・追加	「自走運転システム」では目的地に到達するという目的が明確で現在の位置情報はＧＰＳの技術の進歩によりセンチメートル単位で可能であり、センサーなどにより的確な位置と現在の状況把握が可能である。 「ソフトウェア保守」①障害に関しては、事象の確認と原因分析のために調査が必要となるが、人手作業になっている。②仕様変更・追加などのように実際に行う保守のインプットになる内容を詰めていくフェーズが必要となる。
判断	①人工知能	①保守担当者	「自走運転システム」では、自走するためのインプット情報が人工知能に整理蓄積されているかがポイントである。シミュレーション等で解析された自走のための情報と繰返し行う試行テストで人工知能の判断精度を上げている。 「ソフトウェア保守」ではインプット情報に対して、システムが可視化されていないため、インプットの度にシステムの解析と影響分析を保守担当者が行う必要がある。
アウトプット	①目的地到達のための車体操作（ハンドル・アクセル・ブレーキなど） ②インプット情報・判断基準を人工知能データベースに格納	①障害時：障害報告書、プログラム、修復データ等 ①仕様変更・追加時：成果物（設計書、プログラム等）	「自走運転システム」では、目的地到達という目的が明確。インプット情報・判断基準などもデータベースとして活用し、車体を操作する。 「ソフトウェア保守」の場合、ドキュメントとして報告書に纏めるが、アウトプットをデータベース化してナレッジとして可視化や予防保守に役立てているケースは、一部特定されたシステム（パッケージなど）に限られているのが現状である。

３－１－３．ソフトウェア保守の自動化今後の課題

ソフトウェア保守の場合、前項の表３－１「自走運転システムとソフトウェア保守の機能比較」で考察した通り、自動化の道は険しいと言わざるを得ない。

① インプット情報をデータベース化して、原因分析、影響範囲、対応方法を何処まで自動化出来るか？　ＩＴＩＬ（Information Technology Infrastructure Library）などの活用も期待される。

② ①を実現するために、現在属人的で可視化されていない部分を如何に可視化していくか？　今後アプリケーションパッケージは多くのユーザーを抱え、より生産性の向上に取り組まざるを得なくなり可視化が進んでいくと思うが、個別アプリケーションの場合可視化スピードは鈍いと思われる。

③ １０年後のソフトウェア保守自動化の牽引はアプリケーションパッケージで、その進化が今後のソフトウェア保守自動化のスピードに大きな影響を与えるだろう。

３－２．ＩＴ業界の運用の自動化動向

次にやはり最近報道されたＩＴ業界の今後大きな流れとみられる「運用の自動化」とクラウドビジネスも絡めたＩＴ業界の今後の動きに関して触れてみたい。

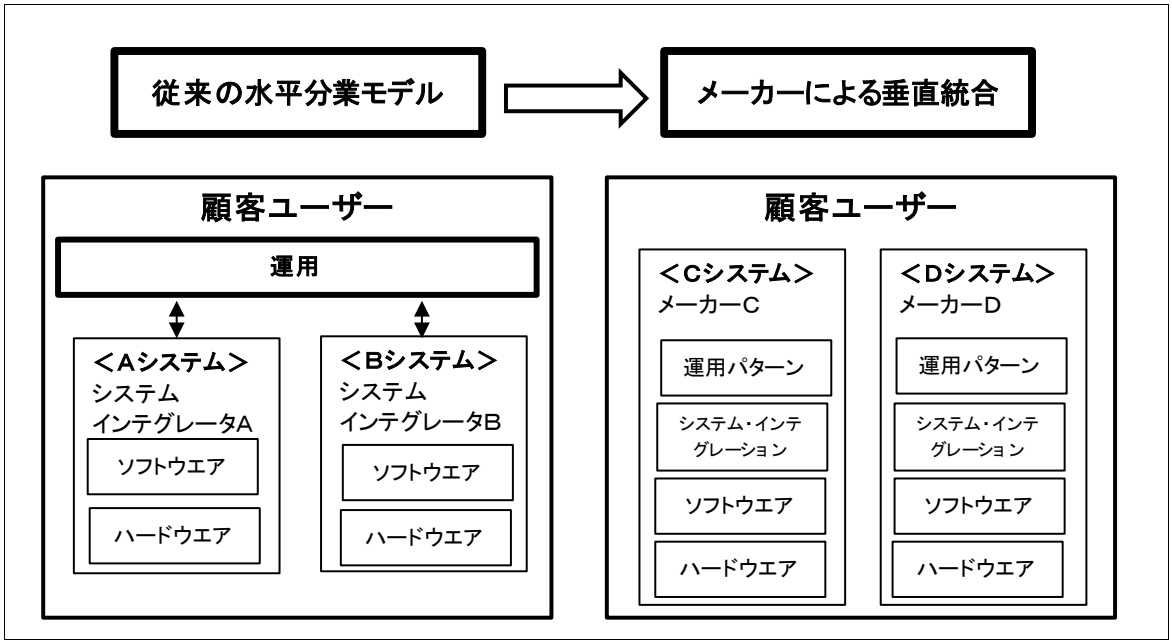
「運用の自動化」を取り上げたのは、システムがサービスインした後、ハードとミドルウェアなど関連ソフトを管理する「運用」と、アプリケーションソフトウェアなどを管理する「保守」は車の両輪であり、その片輪の「運用」の自動化に関してその動向は、ソフトウェア保守の将来を議論するうえで見逃せない事象と思ったからである。日経コンピュータ６月７日号にその特集が載っていたのでその一部を紹介したい。

３－２－１．水平分業型ビジネスモデルの終焉と垂直統合型ビジネスモデルの台頭

２０年以上行っていたシステムインテグレータが、ハードウェアとソフトウェアを組み合わせ、それぞれのメーカーに発注していた水平分業型は将来的には少なくなる。なぜなら、現在のオープンシステム化により顧客はマルチベンダー構成になり、システムが複雑化し運用費用の増大はもはや許されない状況になっているためである。

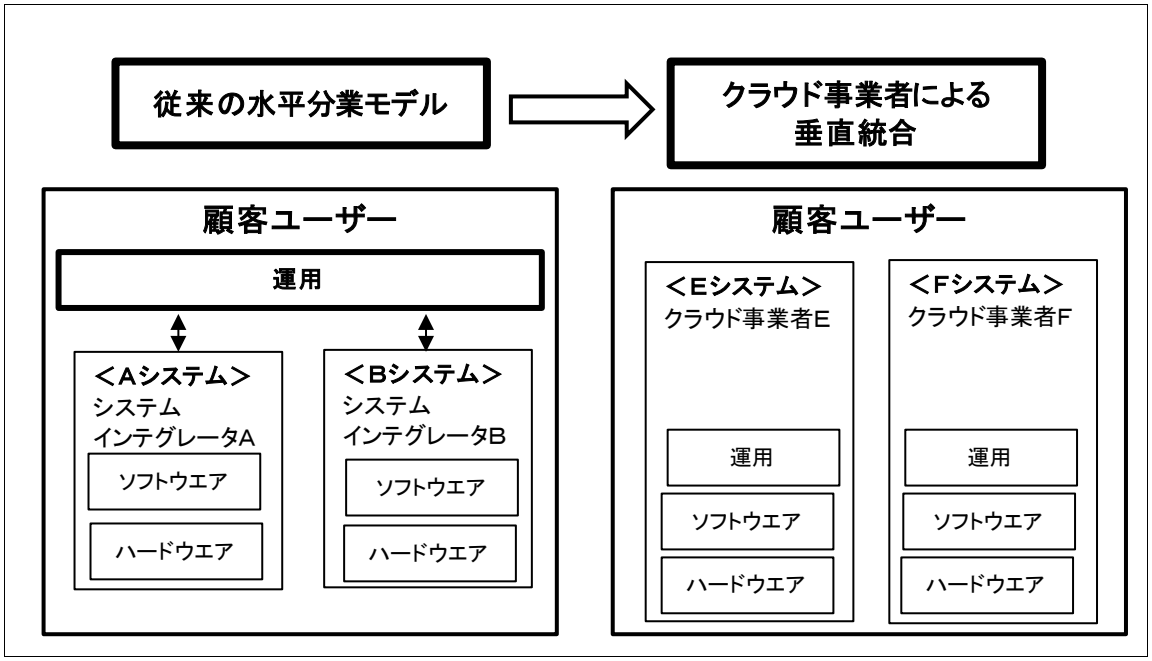
現在の水平分業型に替わり、ハードウェアとソフトウェアを最適化して出荷し、ＩＴ支出の７０％を占めるインテグレーション費用を削除することをセールスポイントにした会社が出現している。他のメーカーでも、インテグレーション費用だけではなく、人手による運用を不要とするシステムを発表した。背景には、今までに蓄積した運用ノウハウをパターン化し、運用管理ソフトとして自動化を実現可能したことによる。運用人件費の８６％削減を可能とすることを最大のセールスポイントとしているようである。

図３－２　メーカーによる垂直統合



３－２－２．もう一つの流れクラウド型

図３－３　クラウド事業者による垂直統合



クラウドは、ユーザー企業に代わってクラウド事業者が運用やソフトウェア保守を行うことで、ユーザーに取ってメリットは大きいと言えるだろう。

しかしながら、クラウドを提供する事業者に取って、ユーザーの抱え込みとしては有効かも
しれないが、一部の大手クラウド企業を除いて、クラウドビジネスの裏側は依然とした人手による運用・保守が続いているのが現時点の実態で、暫くは続くと思われる。

その中で、クラウド事業者の中にはハードを製造して垂直統合型に移行していく会社も今後出てきて、１０年後はそのような垂直統合技術を持たないＩＴ企業の淘汰が進んで行くと思われる。

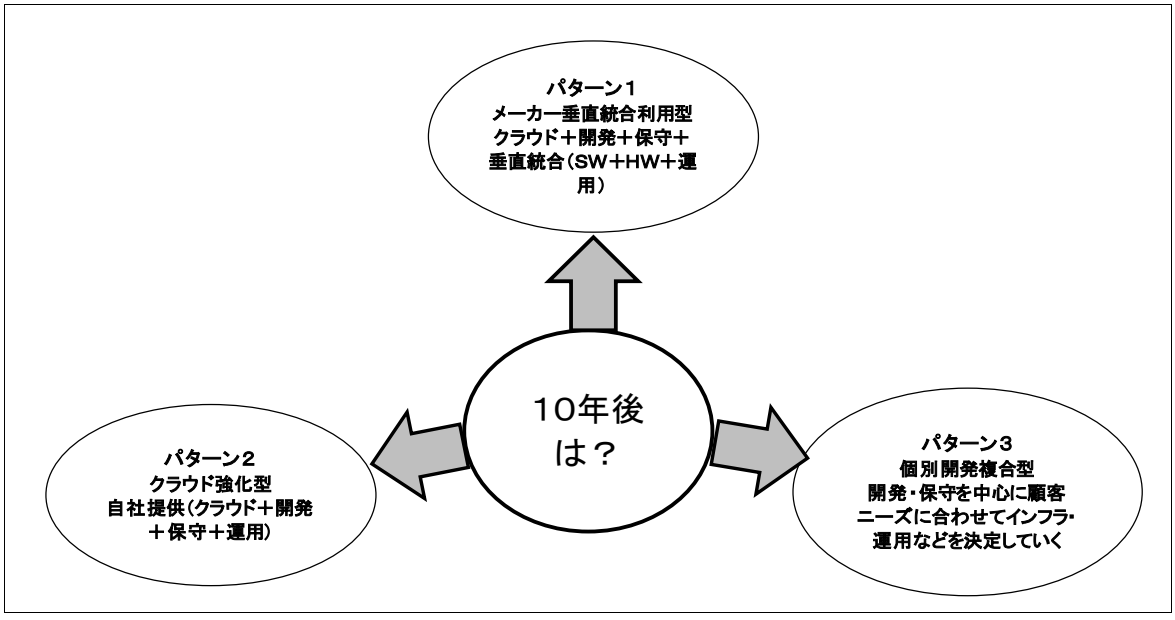
３－３．ＳＩビジネスの１０年後とソフトウェア保守は？

今年度のテーマであるソフトウェア保守を含めたＳＩビジネスの１０年後はどうなるのだろうか？

今後、クラウドビジネスが浸透していくのは目に見えている。クラウドの出発点も各社の業務が似ているグループウェアからであり、戦略的システム以外のアプリケーションパッケージなども今後クラウドビジネスとしてより進化していくだろう。

前節の「運用の自動化」を踏まえ、今後ＳＩビジネスは次のパターンの三極化が進むと思われる。

図３－４　ＳＩビジネスの１０年後



３－３－１．パターン１：メーカー垂直統合利用型

パリーン１とは、メーカー独自の垂直統合（ソフトウェア＋ハードウェア＋運用）利用を前提にしたビジネスモデルである。パッケージソフトウェアも含めたアプリケーションの開発と保守をセールスポイントにしていくメーカー企業は世界中でもそう数が多い訳ではないが、今後企業買収をより進め巨大化していくであろう。

ポイントは、メーカーの垂直統合を利用し、尚且つパッケージソフトウェアなども充実させ、メーカーが得意とする垂直統合型とセットで売り込むケースも多くなっていく。パッケージを充実させるためメーカーは、企業買収や企業提携がより一層多くなり、メーカーの顧客囲い込みが進むだろう。

また、メーカー独自のクラウドと自社内運用で垂直統合を使用してサービス提供するビジネスモデルも用意し、フルアウトソーシングと同様だが低価格のサービスビジネスも今後拡大していくだろう。

３－３－２．パターン２：クラウド強化型

パターン２とは、自社提供のクラウドサービスをベースに、アプリケーション開発・保守と運用をすべて行うビジネスモデルである。この場合、サービス提供はメーカー企業ではなくクラウド事業者なので、垂直統合（ソフトウェア＋ハードウェア＋運用）を行っているかは顧客にとっては見えない部分（ブラックボックス）であるが、クラウド事業者にとっては、運用や保守の生産性向上を行わないと生き残りは難しいと思われる。

このモデルの特徴として、クラウド事業者には①から③があり、それによりサービス内容が少々異なると思われる。

①ＳＩベンダー：PaaS（Platform as a Service の略、パースまたはパーズ）

アプリケーション開発を行い、クラウドを利用して運用・保守サービスを実施する。

②パッケージベンダー：SaaS（Software as a Service の略、サース）

パッケージを開発し数多くのユーザー顧客に提供し囲い込みを行うため、クラウドサービスを提供。メーカー垂直統合利用型サービス提供企業もクラウドサービスを行っており、今回補足として記載する。

③データセンターベンダー：IaaS

（Infrastructure as a Service の略、イアース,アイアス）

ハードウェアなどインフラと運用サービス提供を行い、アプリケーションソフト保守はユーザー顧客またはＳＩベンダーが実施する。

ソフトウェア保守に関しては、上記①と②のベンダーがサービスを提供する。どの程度の自動化が出来るか、各クラウド企業の生産性向上結果によると言わざるを得ない。
ただし、②パッケージベンダーは多くのユーザーを抱え、ソフトウェア保守の自動化に向けた取り組みが期待される。

３－３－３．パターン３：個別開発複合型

パターン３とは、アプリケーションを個別に開発・保守していく個別開発モデルである。現在ＳＩベンダーが請け負っているビジネスモデルの発展形。インフラ部分（ハード、ソフト）や運用などの要件は、ユーザー企業とＳＩベンダーで決定していく。

顧客にとって基幹となる戦略的システムなどを構築・運用・保守を行うケースで、このパターンは１０年後も続いていると思われる。戦略的システムは今後もＥＲＰなどを中心にユーザー企業にとって生命線となるシステムのため、一般的なクラウドサービスを利用することは考え難い。

ＳＩベンダーに取って、どこまでの範囲をビジネスとして受注できるかがポイントとなる。クラウドビジネスの出現で、アウトソーシングビジネスとの違いを別途定義しなければならないが、クラウドビジネスとアウトソーシングビジネスの比較（機能・価格）が今後話題となると思われる。

ＳＩベンダーは、アウトソーシングも含めた受託サービスとクラウドサービスとの価格比較をユーザー企業から求められるため、少なくともメーカーが構築した運用の自動化のための垂直統合利用型サービスを利用するなどして、生産性向上の取り組みは最低限必要と思われる。

本題であるソフトウェア保守に関しては、個別開発で当ビジネスパターンが継続する限り、残念ながら１０年後も現在と同様な属人的保守が続くと思われる。

（玄間　稔）

４．１０年後の保守環境(デバイスフリーがもたらす保守環境への影響について)

４－１．デバイスフリー端末の利用

昨今、スマートフォンやタブレット端末の爆発的な普及は個人利用のみならず、その機動性、利便性を活用する企業が増えてきている。

今後もその勢いは衰えることはなく、あらゆる分野の企業システムでその活用が拡大していくと思われる。

また、企業がスマートフォン、タブレット端末を採用することにより、それらのデバイスに特化した独自のアプリケーションを開発し利用しており、ソフトウェアの保守現場ではそれらスマートデバイスとそのアプリケーションを管理する必要性が高まっている。

私が保守を委託されている企業システムにおいても、スマートデバイスの試験的導入が進んでおり、メール、グループウェア、営業支援ツール、ナレッジの共有などに独自で開発または、パッケージ製品が提供するスマートデバイス向けのアプリの活用を始めている。保守の現場においてもそれらスマートデバイスをユーザーから借り受け保守を行っているが、アプリケーションは各々の開発元ベンダーが開発を行ったものを受け入れているため、スマートデバイスはエンドユーザーからの問い合わせ時に動作を確認する端末としての利用にとどまっている。

企業が営業支援ツールなどとしてスマートデバイスの利用を拡大する一方、保守の現場においてはその利用は進んでいない。しかし、今後保守の現場においてもスマートデバイスを活用するシーンが増えていくと考えている。

現在の保守現場の環境を捉えながら、スマートデバイスが保守の現場において有効活用できるかどうかを考えていきたい。

４－２．保守の現場における現状の環境

スマートデバイスがどのように有効活用できるかを考える前に、現在私が保守を委託されている企業の業務システムの保守環境を以下に整理し説明させていただきたい。

はじめに、システム、プラットフォーム、保守担当者数、作業場所について表４－１「保守環境」に示す。

表４－１　保守環境

システム	プラットフォーム	保守担当者数	作業場所
フロントオフィス	汎用機	3名	自社
バックオフィス	汎用機＋オープンシステム	4 名＋開発ベンダー（９名）	自社＋開発ベンダー先
情報系	オープンシステム	3名＋開発ベンダー（２～３名）	自社＋開発ベンダー先

現在から 5 年程前までの保守環境は、すべて汎用機上で各業務システムが稼働していたが、業務システムの老朽化に伴い段階的にシステムのリプレイスを重ね、汎用機のシステムからオープンシステムへ切り替えを図っている。但し、フロントオフィスシステムと一部のバックオフィスシステムにおいては旧来から使用している汎用機のシステムを現在も継続利用している。

また、各業務システムのリプレイスは、我々アウトソーシングベンダーが提案し導入を進めたパッケージもあれば、他のベンダーがコンペで勝ち取り、導入を進めたパッケージもある。システム導入後の保守について一元管理したいというユーザー要望により、保守業務は、我々アウトソーシングベンダーで引き受けている。

我々保守のアウトソーシング先ベンダーとしては、このような他ベンダーが導入したシステムを維持管理していかなければならないが、各種保守対応は基本的に導入ベンダーと我々が結ぶ製品保守契約、アプリケーション保守契約に基づき導入ベンダーへ保守作業の依頼が発生している。我々の保守業務には、ユーザーから受け付けた問い合わせ、変更依頼に対し、各開発ベンダーへその作業指示や進捗管理、各種問い合わせの橋渡しなど、開発ベンダーの管理が保守業務に含まれる。

一方、従来から残る汎用機上の業務システムに対する保守は、リプレイスを経てオープンシステムに切り替えが進んだ結果、その保守範囲は縮小している。

保守範囲の縮小に伴い、汎用機のみを使用していた数年前と比較するとその保守費用は３０％ほど削減されている。

また現在も継続したコスト削減努力を要求されているが、システム延命に伴いエンドユーザーからの改善要望は増加傾向にあり、限られた人員で多くの保守作業を行わなければならならない状況にある。

４－３．現状の保守現場の問題点

このような保守の現場においてデバイスフリー端末導入により改善が想定される現状の問題点を挙げてみたい。

① 保守担当要員の不足と属人化

５年前までは、現在よりも多くの保守担当者が業務を行っており、各業務システムにおいて最低２名の保守担当者が確保されていた。

主担当者が不在であっても、副担当者が保守作業を進めることが可能であった。

しかし現状は、保守費用削減に伴う保守担当者の整理が進み、担当者１名のシステム守備範囲が拡大し、属人化も一層進んでいる。

副担当者を育成する必要があるが、その副担当者においても他の業務システムを担当しているため、各保守担当者の負担も増加している。

現状、限られた保守担当者で保守を行う場合、そのシステムの主担当者が不在時に他の担当者が単独で作業を実施することは困難であることが多い。

システムの担当者が不在時に障害等が発生した場合、対象システムによってはその主担当者の判断を仰ぐ必要があるため、対応レスポンスが以前と比べて低いこともある。

また、１名で保守を行っているシステムの担当者によっては休暇を取ることを憚ることもありストレスも高くなる傾向にある。

属人化を解消するための教育により他の担当者のスキル向上対策は必要であるが、熟練した担当者と同等のスキルを習得するには時間がかかるため、

② 大量の紙を使用するレビュー

保守開発における各工程のレビュー時には、提案書、設計書、テスト計画書、テスト結果報告書などドキュメントを人数分印刷し、紙によるレビューを行っている。

レビュー時のドキュメントは多いときには１部１００ページほどにも達することがあるが、その大半の資料はレビュー後廃棄されるため、紙資源の無駄遣いが発生している。

ユーザーとの各種打ち合わせを行う際にもそのドキュメントは紙を印刷し持ち込んでいるが、ユーザーからの多様な質問に対応できるよう配布資料とは別に補足資料などを持ち運ぶ場合も多い。しかし、打ち合わせの場に持ち合わせていない資料を参照しなければならない時もあり、即座に対応が出来ずに持ち帰りになることもある。

③ 夜間、休日の緊急対応時

システム停止が及ぼす業務インパクトの高いシステムを保守している担当者は、問題発生時に迅速に対処することができるよう自宅にお客様環境にアクセスできるノートPCを所持しているが、外出する度にノートPCとデータ通信カードを持ち歩くことは困難であるため、緊急時に手元にPCが無く、対応が困難な状況もまれに発生している。

急を要する対応時などは主担当者が持つスキル、ノウハウが必要であり、不在の主担当者がその場にユーザー環境にアクセスできる端末が無い状況下であれば、電話で状況を確認し指示を行うなどの対応を行っている。

４－４．問題点を解決するためのデバイスフリー端末の活用

上記の保守現場の問題点を整理してみた。

① 保守担当要員の不足と属人化

費用削減の流れの中保守担当者を増員することはできない。主担当者不在であっても緊急の場合はその主担当者に質問、相談をしたい。

② 大量の紙を使用するレビュー

印刷する手間もあり、紙を使用したレビューは避けたい。しかしレビュー時の指摘事項をメモする場合などには手書きができる紙でのレビューの方が効率が良いとの見方もある。

③ 夜間、休日の緊急対応

緊急対応が必要な状況において担当者が迅速に対応できる手段が必要である。

これら３つの問題に対しては様々な改善方法、手法が考えられるが、デバイスフリー端末を利用することで改善できることを考えた。

お客様環境にアクセスできる持ち運びが容易なデバイスフリー端末があれば、主担当者は電話で説明を受けるよりも状況を把握しやすく、また設計書などのドキュメントを参照することができれば、ドキュメントを確認しながら対応することができる。

また担当者は、作業場所を選ばず作業をすることができるため、デスクから離れられない現在の仕事のスタイルから解放され、客先へ出向く、社外の研修に参加するなどその担

当者の活動範囲を広げることができると思われる。
さらに、夜間、休日に緊急対応が必要な場合もデバイスフリー端末を利用しそれを常に携行することができれば、場所や時間を選ばず迅速に対応することができると思われる。

各自デバイスフリー端末を持ち寄ってレビューを行うことにより、レビュー時に使用している膨大なドキュメントを印刷することなく、紙資源の削減とその印刷にかかる手間を省くことができる。
紙資源の節約という観点では、ノートパソコンを持ち寄ってのレビューや、プロジェクターを利用し投影した資料を見ながらのレビューがある。デバイスフリー端末では、レビュー対象のドキュメントに手書きのコメントや、マーカーで線を引くなど、紙の資料と似た作業が可能なため、レビュー参加者は、デバイスフリー端末を利用するメリットがあると考える。

４－５．まとめ

現状３つの保守の問題点についてデバイスフリー端末を利用し改善することについて改善効果が期待できそうであるという結論に至った。
しかし、現在のデバイスフリー端末を見ると、保守作業に活用する上でクリアしなければならない問題点もあると考える。
各システムへデバイスフリー端末でアクセスし、エミュレータやアプリケーションの動作確認や作業がどこまでできるかという問題や、緊急対応とはいえ保守作業を物理キーボードの無い端末で行うことで、オペレーションミスが発生する頻度が高くなる懸念などが考えられる。
上記のような問題点を解決しなければ現実的に保守業務に活用することは難しいかもしれない。
しかし、デバイスフリー端末の進化とともに、１０年後の保守現場においてもその活用が広がり、保守の仕事のスタイルも変化していると考える。

（丸山　陽一）

５．私が考える１０年後のソフトウェア保守

５－１．はじめに

１０年後のソフトウェア保守はどう変わっているのか？その上で今、ベテラン・エンジニアが若手エンジニアに、何をどのように伝承して行けば良いか？

前者の問については、私の過去の経験から１０年後のソフトウェア保守を考察してみた。結果、「ソフトウェア保守技術はあまり変わらず多様化・複雑化している」と言う結論に至った。既存のコンピューター構成要素の 1 つであるソフトウェアは、ハードウェアのように保守切れで破棄されるのではなく、改修と追加を続けているからである。また、技術の発達に伴い、新しいツールや技法も登場してはいるが、古いソフトウェアの保守までカバーするものは見当たらないからだ。ビジネスの進化に対応するために、新旧のソフトウェアに手を入れることが常に発生し続けている。従って、１０年後もソフトウェア保守技術は多様化・複雑化して残っていると考えた。

後者の問については、前者の結論を前提として考察してみた。結果、「ベテラン・エンジニアが持っている経験・知識・技術・技法を伝承して行く必要がある」と言う結論に至った。“何を“については、伝承すべきものごとを洗い出し、“どのように“については、「棚卸」「取捨選択」「伝承」というステップで進める方法を考えた。

５－２．過去から現在

（1）１９８０年代

前半には、メインフレーム中心で開発を行なっていた。外国製メインフレームから国産製メインフレームへ乗り換える企業や大学、省庁が相次いでいた。国産製メインフレームはハードウェアの費用が格安であったこと、日本語処理が得意であったことが普及の理由であった。PL/I や COBOL、アセンブラの資産を移行する仕事を多くこなした。DBMS が登場したことによる情報のデータベース化や、データの日本語化による画面や帳票の変更、作り直しも行なっていた。

保守開発ツールは乏しく、設計書は全て手書きであり、図形やフローチャートを書くための定規が必須であった。設計工程は自社や顧客のオフィスで行なうことができたが、製造・テスト工程は、メインフレームのある場所で行なっていた。メインフレームと直接繋がっている DAM 端末で開発を行ない、何人かで DAM 端末を共有して製造・テストに当たっていた。テスト結果については、メインフレームと直接繋がったプリンターから出力された、レコードのダンプリストで検証を行なっていた。TSS を使ったキーボード中心の作業であった。

後半では、自治体のクライアント／サーバーシステムのソフトウェア開発を経験した。クライアント／サーバーシステムと言っても、サーバーはメインフレーム、クライアントはワークステーションであった。OS やミドルウェア、保守開発ツール、開発運用標準に至るまで、全てメーカー独自のものであった。保守開発ツールは、プログラム設計書は YAC や HIPO（いずれも専用定規使用）、基本設計書や内部設計書はワープロ専用機で作成していた。当時はワークステーションと言われる UNIX 機を使って、製造・テストを行なっていた。キーボードの他にマウスやフロッピーディスクが追加となった。

（2）１９９０年代

前半には、自治体と通信業者のメインフレームやクライアント／サーバーシステムのソフトウェア保守を経験した。パソコンの企業への普及(DOS から Windows)や、サーバーOS の多様化(Windows NT、OSS の Linux、UNIX など)により、保守言語は VB や C が追加になった。保守開発ツールについては、メインフレームでは YPS や YPS/APG（YPS based Application Program Generation system プログラム自動生成ツール）、パソコン上で COBOL のコーディングから単体テスト迄ができるツールが保守開発業務に導入された。企業の各部門へのクライアント／サーバーシステムの導入や、パソコンの利用者への普及に伴い、OS の設定変更、ソフトウェアの入替、利用者への教育など、ソフトウェア保守業務以外の作業も行なっていた。

後半では、製造系のメインフレームのソフトウェア再構築を行なった後、商社系システムの 2000 年問題(Y2K)対応を経験した。下流工程からの対応であったが、商社の色々な部門の業務とシステムを知ることができた。Y2K には、ソフトベンダーのソースコード分析ツールを導入して対応した。但し、分析ツール実行結果による対応要否については、ソフトウェア保守技術者が判断する必要があった。テストもモジュールテストだけでなく、システムテストまで行なう必要があった。当時はパソコンでのソフトウェア保守開発が当たり前になっていた。Word や Excel、Visio などでドキュメントを作成するようになった。パソコン通信が普及し、顧客とメールでやりとりが出来るようになった。

（3）２０００年代

前半には、商社系システムのソフトウェア保守を続けて経験した。EDI などによる企業間連携を行なっているシステムが多く、他企業の同業他社エンジニアとの会話も増えた。時価会計導入、基幹システムの SAP 導入対応、リホストと言われるメインフレームからサーバーへのシステム移行提案など、顧客システム部門と一緒になって行なっていた。保守ツ

ールとしては、グループウェアが協力会社も含め全員に導入され、情報の共有化が図られた。帳票の電子化も一部導入され、紙ベースのドキュメントが一時期少なくなった。

後半では、損保系代理店システムの Web オンライン、メインフレームと各サーバーを繋ぐ CORBA 通信ミドルウェアのソフトウェア保守業務に従事することになった。サーバーの種類も Web サーバー、データベースサーバー、HUB サーバー、ビジネスロジックサーバーなど、様々なコンピューターがインターネット網で繋がるようになった。サーバー側システムの機能拡張により、インターフェース追加絡みのミドルウェア変更を行なってきた。

2008 年にはＩＴ全般統制により、アクセスコントロール、開発と運用業務の分離、保守作業の証跡などのログ取得追加、移行時のコンテンジェンシープラン作成ルール化など、セキュリティやリスク管理が強化され、今まで顧客と一緒に効率化を図ってきた、ソストウェア保守作業手順が通用しなくなった。保守ツールとしては、サーバー側についてはソフトベンダーの C 言語対応フレームワークが開発標準として導入されており、構成管理や障害管理の自動化・省力化を実現することが出来るようになった。

（4）現在

現在は、企業の合併に伴い、更なるシステム連携が増加し、ＳＬＡの１つであるレスポンスに悪影響が出る事象が発生するようになった。この問題を改善するために、業務アプリの保守チーム（メインフレーム側やサーバー側）に対し、事象発生の要因調査と原因の特定を行ない、データベースのチューニングやプログラムのロジック、運用の見直しを顧客システム部門と一緒になって行っている状況である。

直近では、ｉＰａｄ対応で新たなフロントシステムの開発が行なわれている。また、合併に伴う統合新システムの開発が進んでおり、更にシステム連携が増加することになる見込みである。

５－３．１０年後の予測

１０年後のシステムやコンピューター、ソフトウェアの環境はどうなっているのだろうか。その時のソフトウェア保守環境はどう変わっているか。前節での私の経験から未来を考察してみた。

２１世紀に入り、色々な「もの」や「こと」が大きく変化する、激動の時期を迎えている。５０年ほど前にコンピューターが世の中に登場し、コンピューターは色々な分野に普及した。現在では、コンピューターは電気・ガスや水道のようなインフラとなり、コンピューターなしでは事業や生活が成り立たない状況になっている。コンピューターの構成要素の１つであるハードウェアは「ドック・イヤー」と言われながら進歩してきた。もう１

つの構成要素であるソフトウェアについても、ビジネスの変化やハードウェアの進化に歩調を合わせて進化してきた。３０年ほど前は、ＦＡとかＯＡと言われ、製品の品質向上、ビジネスの省力化、経営の効率化のために、コンピューターが続々と企業に導入されて行った。

２０世紀のおわりには、インターネットの登場によるネットワークの進化に伴い、コンピューター同士がどんどん繋がるようになった。結果、企業や人間同士の情報連携のスピードアップが進み、ソフトウェアが処理するデータ量の増加や、データの質もテキストベースからHTMLやXMLに変化した。

現在は、技術の発達によるコンピューターやネットワークの更なる進化により、企業・人・システムや物がどんどん繋がる時代になる。ハードウェアとソフトウェアはあらゆる物へ普及が進みつつある。

１０年後には、コンピューターの小型化進み、無線技術の発達により、携帯電話やスマートフォン以外の身近なものにもコンピューターが組み込まれると考える。例えば、腕時計のようなデバイスが、人間の生命徴候(バイタル)情報を医療機関のコンピューターと送受信することで健康状態を監視し、異常があれば本人にアラーム音で通知する様になっているのである。また、音声認識技術の進歩で、「あの商品いくらになったかなぁ」とつぶやくと、「今、キャンペーンにつきいくらですよ」と量販店のコンピューターから応答あるようになっている。また、医療分野での介護支援ロボットの導入により、人間やコンピューターとのインターフェース（マン・マシンインターフェース）も、新しい規格や標準が定められるだろう。

次に１０年後のシステム環境、ハードウェア環境、およびソフトウェア環境について述べる。

（1）システム環境

コンピューター技術の進化も速いがビジネス変化も速いため、迅速なシステムの対応も必要になっている。また、国内企業・海外企業との合併や連携が、現在よりも多くなっている。それに伴い、国内外のシステム関連携も増加し、システム対応が複雑になっている。

また、システムを利用する側についても、低年齢層・高齢層・傷がい者・外国人と多様化している。より高度な安全性やセキュリティの確保が、システムリリースの判定基準のひとつになっている。

（2）ハードウェア環境

メインフレーム、クライアント／サーバー、データセンター、インターネットの普及によるサーバーの細分化、更にクラウドも追加され、集中型と分散型が混在することで、コ

ンピューターの形態も多様化が進んでいる。
デバイスも専用端末（ＰＯＳなどの機器）からパソコンへ、モバイル端末（携帯電話、スマートフォン、ｉＰａｄなど）からスマートグリッド、自動車、家電など、身の回りにあるほとんどものにコンピューターが普及する。従って、デバイスの多様化も進んでいる。

（2）ソフトウェア環境

システムが繋がれば繋がるほど、ハードウェア形態が多様化すればするほど、インターフェースの追加変更やトランザクションなどのデータ増加が発生し、ソフトウェアの対応も必要になる。従って、ソフトウェアの保守は、増えることがあっても減ることはない。あらゆる物の中でソフトウェアが動くことになるため、ソフトウェア環境も複雑化する。テキストベースのデータ処理の他に、音声や画像ベースのデータを同時に扱うようになっている。データサイズも増加するに違いない。ビックデータや多様なデータ（マルチメディアという言葉が過去にはあった）を高速に処理出来るソフトウェアも必要になる。従って、ソフトウェアを作成するときの言語も多様化している。リレーショナルデータベースだけではなく、新しいデータベースシステムも開発されている。それに伴い、保守開発ツールやテストツールも進化して多様化している。

５－４．１０年後のソフトウェア保守技術者

１０年後のソフトウェア保守技術者には、ソフトウェアの多様化・複雑化が進むことで、今より幅広い知識や多くの技術が必要となる。その中で、ソフトウェア技術者はどうなっているのか、どうなっている必要があるのか考察してみた。

（1）保守プロセスの中での専門化（分業化）

多様化・複雑化が進めば、１人で保守業務を全てこなすことは難しくなる。１人で面倒を見る範囲を越えてしまうからだ。システムがどんどん繋がり、あらゆる物にソフトウェアが浸透した世界では、ひとつの組織や企業で対応するにも限界が出で来る。結果、ソフトウェア保守業務の専門家（分業化）が進むと考える。ソフトウェア保守プロセスには、「問題把握及び修正分析」「修正の実施」「保守レビュー及び受入」「廃棄」のアクティビティがある。このアクティビティ毎、あるいはアクティビティ内で分業、あるいは職業が分かれるのではないか。例えば、「問題把握及び修正分析」については、システムを利用する企業を横断して調査を専門家が行なう。「修正の実施」については、ソフトウェア改修は言語の多様化により、専門の会社が行なう。と言うことが一般化しているのではないかと考える。

（２）経営的センスが必要になる

ビジネスの進化によるシステム変更要件や新規要件が発生した場合、既存のソフトウェアを改修した方がコスト的に有利なのか。スクラッチ開発することで初期コストはかかるがランニングコストで有利になるのか。顧客のＩＴ投資における費用対効果判断する、経営的センスを持った人材か必要になる。

課題や問題が発生した場合、現在動いているソフトウェアが今後どうなるか見極めて判断する能力も必要になる。どこのソフトウェアを修正すべきか、テストはどう進めて行けば良いのか。適切なコンサルテーションが現場層や経営層に出来る人材が必要となる。あまり利用されないソフトウェア、利用者数が少ないソフトウェア、長年安定稼動している古いソフトウェアに対し、対応することで保守コスト増になる場合は、ソフトウェア保守を止める手もある。「長い目で見た結果、保守はもうしない方がよい。ユーザー側のパソコンでこう処理すれば要件は満たせる」と提案した方が、顧客側が負担するコストを下げることになる場合もあるからだ。

５－５．若手エンジニアへの伝承の必要性

ソフトウェア開発では、一定の期間内にシステムを構築するため、各専門家（プロジェクトマネージャ、アーキテクト、データベースエンジニア、ネットワークエンジニア、アプリケーションエンジニア）の技術や知識が要求される。ソフトウェア保守ではそれに加え、システム運用に係わるリソース（人・金・物・情報）を恒常的にフォローして行く技術や知識が追加で必要となる。

ソフトウェア保守の現場では、経済の低迷による企業のＩＴ投資の減少により、十分なリソースを与えられずに業務を回しているところが多いのではないか。ビジネスとシステムの進化の中で、ベテラン・エンジニアの守備範囲はどんどん広がるばかりである。その様な環境では、ベテラン・エンジニアの持つ知識や技術、カンやコツの若手エンジニアへの伝承は後回しとなる。

その様な状態のまま世代交替した結果、保守現場の生産性や品質が低下したことが原因で、保守によるトラブル発生に陥る可能性は十分あると想定する。ベテラン・エンジニアから若手エンジニアへの伝承は、今後１０年間で必要性が高まると考える。

そこで、若手エンジニアへ何をどのように伝承するのか手法を考えてみた。

５－６．若手エンジニアへ伝承すべきものごと

ビジネスやシステムの変化は過去よりも速くなっているため、若手エンジニアは、ＩＴ業界の新しい技術や技法を習得して行かなければならない。この状況で更に、ベテラン・

エンジニアの豊富な経験や膨大な知識を受け入れようとすると、若手エンジニアが新しい知識や技術を覚える余裕（伸びシロ）がなくなってしまうことになりかねない。また、私たちはエンジニアでもありサラリーマンでもあるため、職人や芸能家のように何十年もかけて伝え続けることは、組織的・時間的・コスト的にも難しいと考える。効率的に伝えて行くためには、どのようなものごとがあるか考えてみた。次に「システム・ソフトウェア進化の歴史」「顧客（組織）の文化」「保守・開発・運用プロセス」「各種スキルとエンジニアマインド」「システム・ソフトウェア構成要素」「業務知識（ソフトウェア）」について述べる。

（1）システム・ソフトウェア進化の歴史

ソフトウェア保守の現場には、○○システムやサブシステム、○○業務と呼ばれているシステム全体図があるはずだ。オンラインやバッチなどのシステム運用形態で分けて作成されている場合もある。このシステム全体図をもとに、システムやソフトウェア進化の歴史を６Ｗ４Ｈで補足し伝える。表５－１「６Ｗ４Ｈの内容例」を次に示す。

表５－１　６Ｗ４Ｈの内容例

英語	日本語	内容例
Who	誰が	構築した組織や企業、保守している組織や企業
Whom	誰に	利用している組織や企業、消費者など
When	いつ	サービスを提供している時間（オンライン・バッチ） サービス停止タイミングや時間も含める
Where	どこで	ソフトウェアやハードウェアのある場所
Why	何のために	社内事務の効率化、販売促進、コスト競争力強化など
What	何をするのか	会計管理、販売管理、在庫管理、営業支援、 オンライン制御などシステムの機能概要 連携システムとのインターフェイス情報
How to	どの方法で	ＯＳ、ミドルウェア、アーキテクチャなどシステム構成 要素、開発ツール、言語、開発手法
How long	どれ位の時間で	開発期間、稼動期間
How many	どれくらい	システムの規模（資源のボリューム）
How mach	いくらで	開発費用、保守費用（毎年法改正で保守が発生）

（2）顧客（組織）の文化

顧客の業務内容、顧客の組織体制と各組織の役割、業務用語、業界用語、顧客や組織の仕事の進め方、組織や人の関係、業界ルールを伝える。お金の決裁権限を持っている人、現場の信頼が厚い人、業務を熟知している人、影のまとめ役などのキーマンや人間関係についても伝える。

（３）保守・開発・運用プロセス

顧客やシステム形態にあったソフトウェア保守手順（プロセス）と運用を伝える。保守プロセス自体はこれからもあまり変わらないが、より重要度を増すと考える。従って、若手エンジニアには、共通フレーム２００７やＳＬＣＰ、ＩＴＩＬ、ＰＭＢＯＫのことも学習するように、ベテラン・エンジニアは機会を見つけて働きかける必要がある。

（４）各種スキルとエンジニアマインド

各種スキルにはテクニカルスキル、ヒューマンスキル、コンセプチュアルスキルがあると言われている。テクニカルスキルについては、ＩＴ業界特有のスキルのことであり、ITSSやETSS、UISSを参照されたい。ヒューマンスキルについては、ＩＴ業界特有のスキルではなく、一般社会人としてのスキルのことである。ヒューマンスキルの中にコミュニケーションスキルや交渉力が含まれる。コンセプチュアルスキルについては、「物事や課題を概念化し、本質を捉える能力のこと」と言われているもので、コンセプチュアルスキルの中に問題解決能力が含まれる。スキルについて定義しだすときりがない。従って、若手エンジニアの立場や得て不得手を見て、段階を追ってレベル分けをして伝える。

エンジニアとしての心構え（物事の考え方）も伝える必要がある。システムやソフトウェアの特性、顧客の文化に沿った形で伝える。若手エンジニアの気質を見て、精神的な病気にならないように、自らモチベーションを維持できるように、どう業務に取り組んだら良いかアドバイスを行なう。従って、エンジニアマインドも伝える。

（５）システム・ソフトウェア構成要素技術

ＯＳ、ミドルウェア、言語、DBMS、オンライン・バッチ処理方式、システム間インターフェース、フレームワークなどのツール群の進化について伝える。

本番環境や保守（テスト）環境で稼動しているソフトウェアについて伝える。２：８理論にもとづいて、頻繁に保守が発生するソフトウェアから伝える

ハードウェアの知識もソフトウェア保守に必要であるため、基盤（インフラ）を管理している部署から、ハードウェア構成図を入手し伝える。

（６）業務知識（ソフトウェア）

ソフトウェアがどのようなアーキテクチャで構成されているか、アプリケーションがどのような標準化や部品、ツールを使って作成されているか伝える。顧客のビジネスロジックとアプリケーションロジックについても伝える。基本設計書、外部設計書、内部設計書、

詳細設計書などの既存のドキュメントを使い、ソフトウェア保守業務を行なう中で伝える。ソフトウェアの良いところや悪いところ、課題や問題も含めて伝える。以下に、設計書以外に伝えることが必要なものを本番環境と保守環境に分けて列挙しておく。

①本番環境
・保守記録、障害記録、見積書
・リリース手順書、リリースツールの利用手引き
・リソース（ライブラリ、ソースやオブジェクト、ＪＣＬ、スクリプトや定義体）

②保守（テスト）環境
・テスト計画書、テスト設計書、テスト仕様書、テスト結果報告書
・保守開発ツールと利用手引き
・テスト手順
・テストデータ

５－７．伝承のためのステップ

前節で説明した６つのものごとについて、どのように伝承して行くか考えてみた。「棚卸」「取捨選択」「伝承」というステップを経て、現状のものごとを正確に伝える方法を提言する。以下に、３つのステップを説明する。

（１）棚卸

棚卸については、ベテラン・エンジニアが主導して行なう。自分自身が時間を作り、明文化されていないものごとを箇条書きでもよいので、思いつくまま全て書き出す必要がある。ソフトウェア保守業務を進めて行く中で、ひとつの保守案件が一段落した時に、時間を確保し、前節で取り上げた６つのことをキーワードにして書き出す。大規模システムや複雑なシステムのソフトウェア保守の場合は、ある程度の期間をとって取り組むことになる。

（２）取捨選択

取捨選択については、ベテラン・エンジニアが主導して行う。１０年後、自分が保守してきたシステムがどうなるのかを予測し、必要なものごとを取捨選択する。６つのことについては全て必要であると考えるが、その幅や深さを見極めるのである。簡単に言うと「捨てる技術」となる。このステップを踏まずに全てを伝承しようとすると、若手エンジニア

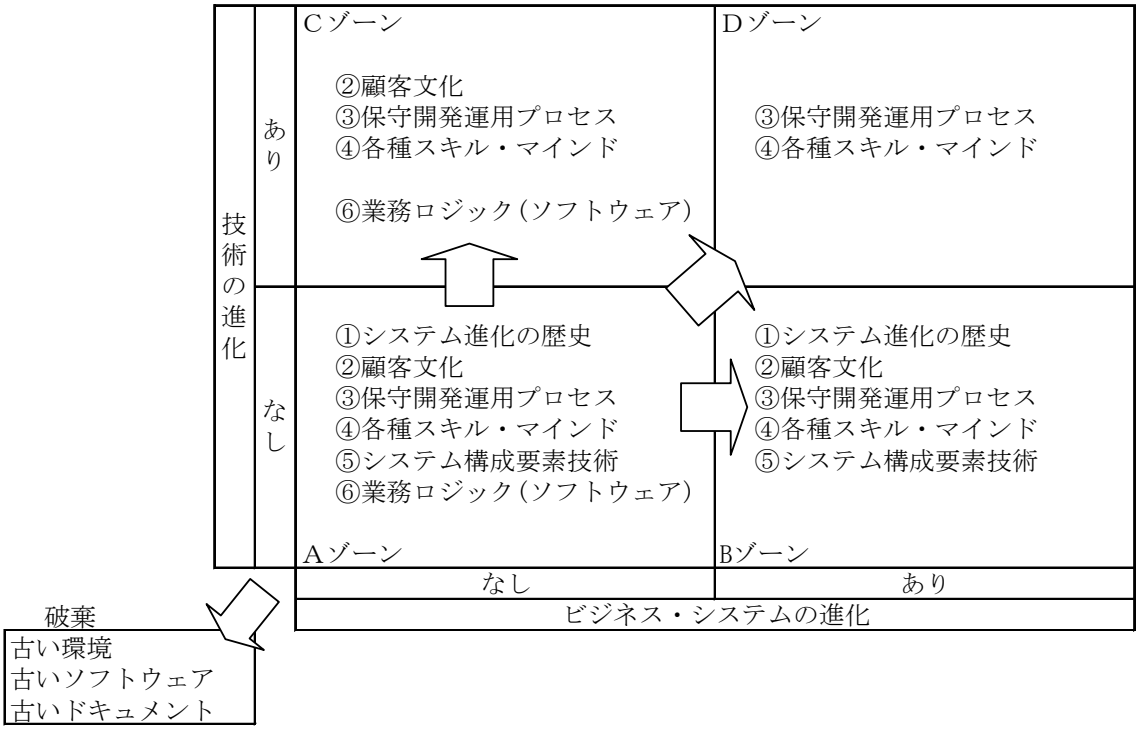
の頭の中が満杯になり、新しい知識や技術を受け入れる余裕がなくなってしまう。

取捨選択を行なうときに、何故そうなっているのかなどの原因を追求しない方が良い。また、起きるか起きないか分からない仕様変更や機能追加も考えない方が良い。取捨選択した結果、使われていないソフトウェア、古いシステム、古いハードウェア、古いテスト環境、古いテストデータが見つかる場合がある。できることならば、若手エンジニアが迷わないようにするために、思い切って破棄した方が良い。

尚、取捨選択を行なう過程で、更新されていないドキュメント、見当たらないドキュメントが出てくることがある。その場合は、最新化や新規作成は行なわない。「伝承」でそのままの状態を伝えるためである。

次に何を伝承するかしないかを決める判断基準の参考（一例）として紹介する。ビジネスが進化すればシステムも進化する。新技術によってもシステムが進化するため、技術の進化とビジネス・システムの進化を軸に、６つのことをマッピングしたものを図５－２「技術とビジネス・システム進化と伝承マトリクス」に示す。伝承するも物事を整理するときの一助として考えてみた。

図５－２　技術とビジネス・システム進化と伝承マトリクス



技術の進化とは、今運用しているシステム全てを新プラットフォームや新技術で再構築や刷新するケースである。メインフレームで運用しているシステムをオープン系システム

に乗せ替えるという場合、クライアント／サーバーで運用していたオンラインシステムを、３階層アーキテクチャを使いWebオンラインシステムに移行する場合を想定している。

ビジネス・システムの進化とは、企業の合併や買収により、顧客が新規ビジネスを始めたケースである。新規にシステムやソフトウェアを追加する場合を想定している。

・Ａゾーンについては、技術・ビジネス・システムの変化がない状態で、今後もソフトウェア保守を続けることを意味している。このような環境では若手エンジニアに、①から⑥全てのものごとを伝承することができる。
・Ｂゾーンについては、既存の技術を使い新システムが追加になった状態で、今後もソフトウェア保守を続けることを意味している。このような環境で若手エンジニアには、⑥業務ロジック（ソフトウェア）以外のものごとを伝承することができる。
・Ｃゾーンについては、ビジネスの変化はないが、新しい技術でシステムを再構築や刷新されていることを意味している。このような環境で若手エンジニアには、②顧客文化、③保守開発運用プロセス、④各種スキル・マインド、⑥業務ロジック（ソフトウェア）を伝承することができる。
・Ｄゾーンについては、新しい技術で新しいビジネスで構築されたシステムになっていることを意味している。このような環境で若手エンジニアには、③保守開発運用プロセス、④各種スキル・マインドのみ伝承することができる。

（３）伝承

伝承につては、棚卸と取捨選択した結果を、若手エンジニアに正確に伝えることにある。設計書やテスト仕様書、調査報告書やトラブル報告書、手順書や規程集については、既に文書があるため、ソフトウェア保守業務を遂行して行く中での OJT に位置づけし、若手エンジニアに場数を踏ませて伝承して行く。文書がないものごとについては、ミーティングのような Off-JT による伝承を行なうことになる。ミーティングには２つの形がある。1 つは、ベテラン・エンジニアがたたき台を作成し、ミーティングで会話しながらブラッシュアップを続ける方法である。1 つは、インタビュー形式でミーティングを行う方法である。若手エンジニアがインタビュワーになり、ベテラン・エンジニアに質問することで得た知識を元に資料を起こして貰うのである。

技術については、今流行りの技術そのものを教えるのではなく、過去から使い続けられてきた技術で、１０年後も残るものごとは伝承すべきである。例えば、言語についても COBOL だ C だ Java だ C#だなど言わずに、手続き型言語とは何か、オブジェクト指向言語とは何か、構造化技法（手法）はどの言語にも必要なのは何故か？などをベテラン・エンジニアの経験を交えて伝承する。その方が若手エンジニア側も、興味を持って受け入れることが出来ると考える。

ＱＭＳでは、要員の教育についても重要な監査事項として位置づけられている。ＱＭＳを導入しているＩＴ企業では、ソフトウェア保守業務でも品質計画書があるところが多い。その中に、ソフトウェア保守業務に必要なスキル一覧とスキルレベルの評価、育成計画がある。スキル一覧に棚卸した項目を追加し、教育の一環として伝承に取り組む方法もある。

５－８．考察を終えて

１０年後の予測について、Google で「過去 現在 未来」をキーワードにして検索をかけてみると、激動の時代には「未来は過去と現在の延長線上にない」というような記事が多くヒットした。ゼロベースでの白紙状態から未来を予想することは、私にとってはとても難しいことであった。しかし、「歴史は繰り返す」と言う諺もあるため、自分の過去の経験から１０年後の予測してみた。

伝承のステップについて、昨年は「断捨離」TM(やましたひでこ)という「物や事の整理術」が世間に大きく広まった。そのキャッチフレーズをもとに考えてみたが、ソフトウェアは物ではなく保守することで進化するため、考え方は非常に参考になったものの、そのまま適用するのは難しいということが分かった。ソフトウェアの修正依頼を絶って、使用頻度の低いソフトウェアを捨て、そのシステムをシンプルに保つわけにもいかないからだ。

定年が６５歳まで延長にはなりそうだが、そこまでベテラン・エンジニアは活躍できるだろうか？　６０歳を超えて体力気力が衰えてくる頃に、さぁ伝承だと言われても、経験や知識がちゃんとアウトプットできるだろうか？　ベテラン・エンジニアは、常日頃から伝承を意識し、少しずつ若手エンジニアに伝えていくのが有効だと考える。

ハードウェアについては近年、５年毎にリプレイスが発生している。ハードウェアの保守期間５年が一般化して来たからだ。ソフトウェアには、ハードウェアのような定期的なリプレイスはない。ソフトウェアについても、保守年数をシステムライフサイクルに取り入れたらどうか。定期的に新システムに切り替わるようになれば、伝承も大きな問題にはならず、伝承の負荷も軽減されると考える。

（伊藤　順一）

６．保守開発のグローバル化：１０年後の国内ＳＩビジネスと保守開発者の心構え

６－１．はじめに

今期のテーマ「１０年後の保守を考える」について、私なりに考察をするにあたり、まずイメージしたことは、現在のプロジェクトで推進しているオフショア開発がどのように変化しているかであった。

中国を主とする低コストの製造工程シフトは、文化やコミュニケーション、為替リスクや現地の賃金上昇といった課題に不透明さはあるものの、エンドユーザーの求める低コストのシステム開発を達成するためには、国内のＳＩビジネスにおいても保守開発のグローバル化が一層進むことは確実であろう。
また、長期に渡り従事している大規模システムのアプリケーション開発においても、大手ＳＩｅｒによるコーディングの自動化が積極的に導入される環境に移行しつつあり、日本の保守開発者にとって１０年後は大変厳しい外部環境となることが想定される。

さらに、従来であれば若手のうちから設計から試験、商用サービス開始に渡り、１人の技術者が幅広く業務を担うことでスキルアップすることができる現場環境があったが、アプリケーション開発手法、ソリューションの進歩により、若手のうちから即戦力として、まずは狭い範囲から実務に入るプロジェクトが増えている。
いわゆる「たたき上げ」の開発者が育つ環境が少なくなっていると感じている。

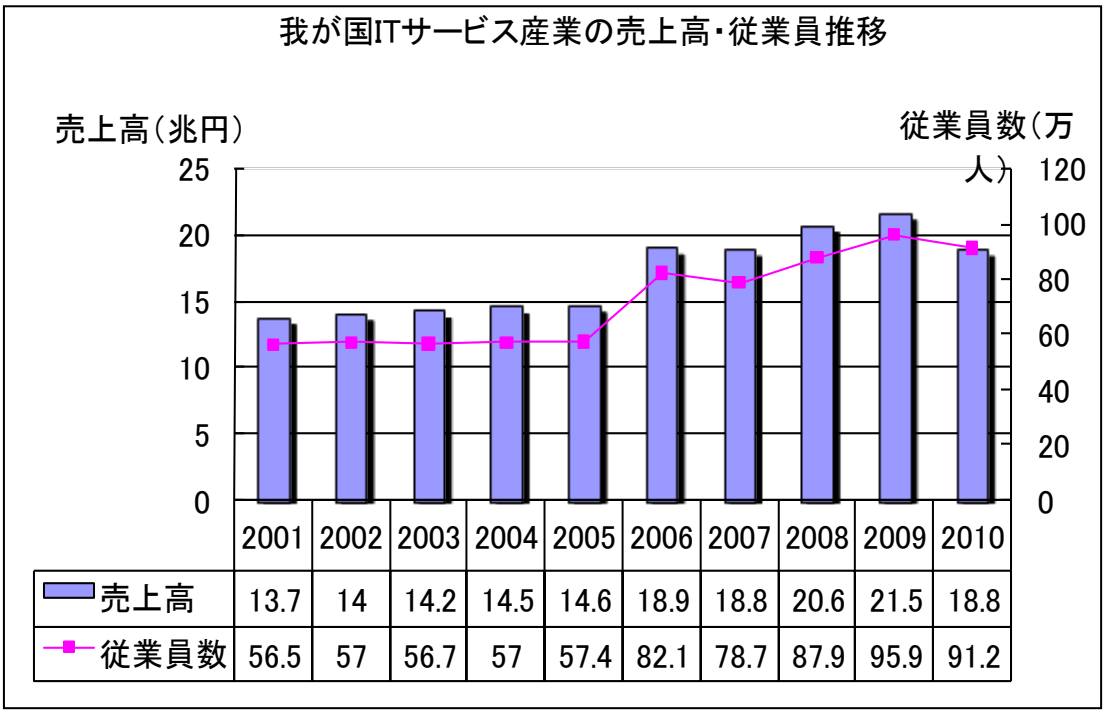
これらを踏まえ、今回の報告では１０年後の保守開発のグローバル化について仮説を立て、その環境に備えるための日本の保守開発者の心構えを提言する。

本論については、下記を参考とさせて頂いた。

- 海外 IT アウトソーシングの進め方とポイント（案）：JISA 国際委員会日中部会
- SI と運用が消える ： 日経コンピュータ 2012.6.7 号
- 激動 トヨタ ピラミッド ： NHK スペシャル

６－２．１０年後の事業環境

参考データ１ ： 我が国情報サービス産業の売上高・従業員推移



	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
売上高	13.7	14	14.2	14.5	14.6	18.9	18.8	20.6	21.5	18.8
従業員数	56.5	57	56.7	57	57.4	82.1	78.7	87.9	95.9	91.2

出典：経済産業省特定サービス産業実態調査

日本のＩＴサービス売上高は、世界的な金融危機を受けたＩＴ投資の減少を受け、直近は低下・横ばいの傾向が見て取れる。今後の傾向として、クラウドに象徴される低コストの統合ソリューションへシフトすることが確実であり、売上高が１０年後に上昇することは考えにくい。

売上高の傾向が、従業員数の減少にも繋がっていることが見て取れる。新卒者の直近の国内ＩＴ業界人気も同様に低下・横ばいであり、少子化とあわせて業界の人員構成も日本全体の人口ピラミッドと同様に少子高齢化で推移するものと予測する。

世界的な観点では、発展途上国のＩＴ化は更に加速するため、ＩＴサービスの売上高は右肩上がりで増加することは必然であるが、日本の大手ＳＩｅｒによるM&Ａも加速しており、国内の開発者が恩恵を受けるというよりは、人材を含めたＩＴサービスのグローバル化が加速すると考えるべきであろう。

６－３．１０年後のシステム環境

「ＳＩと運用が消える」

世界的なハードとソフトを垂直統合したアプライアンスの導入が国内の大手ベンダーでも確実に進んでいる。

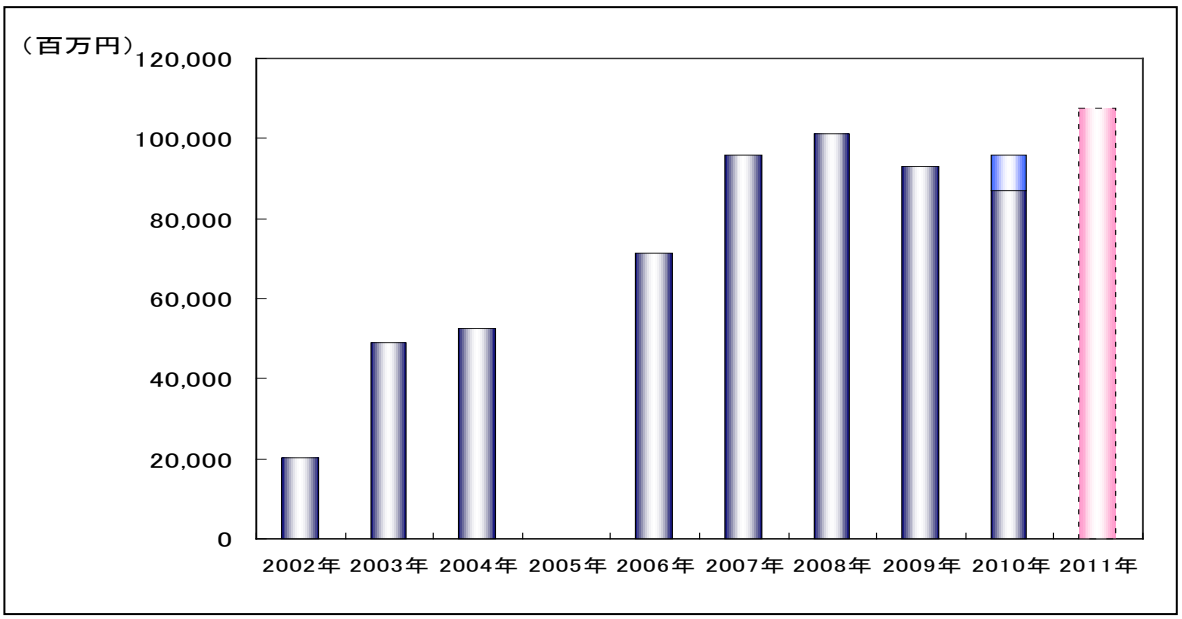
このことは、国内の保守開発者が優位性を持っている運用・保守の暗黙知が自動化され、ニーズと市場が小さくなることを意味しているといえる。

１０年後に想定される高度化された運用・保守環境下では、ベテランのノウハウはシステム化され、さらには若手の実務経験の機会を奪うことにつながり兼ねない。

さらには、自動化されたシステム環境が世界的に広まると仮定した場合、人材としてもコスト・スキルの両面から海外リソースを積極的に活用することが考えられる。現時点でもＢＰＯ（ビジネスプロセス・アウトソーシング）、ＡＭＯ（アプリケーションマネジメントアウトソーシング）として海外リソース・拠点を活用した導入事例も増えていることから、国内企業の保守開発が海外拠点で行われることを想定しておく必要がある。

６－４．１０年後の保守開発グローバル化

参考データ２　：　オフショア発注額の推移



出典：情報処理推進機構「IT 人材白書 2012」

オフショア発注額は日本のＩＴサービス売上高とほぼ同じ曲線を描いている。

この傾向から、様々なリスクを乗り越え、切り出せる部分のオフショア開発がこなれてきたことが見て取れる。

今後進むと思われる賃金格差の縮小の解消策として、国内ニアショア（北海道、沖縄）に戻ることはなく、中国内陸部やミャンマーなどへのシフトを目指す可能性が高く、グローバル化は避けられない。

「激動 トヨタ ピラミッド」

あらゆる業種において下請け企業レベルでの海外進出が確実に進んでいる。
ＩＴ業界においても海外拠点の設立、海外企業のM&Ａが加速しており、ビジネスとしても国内の保守開発者は海外との協働を前提に自らの今後のキャリアを考えざるを得ないといえる。

６－５．１０年後への備え

上記で仮説を立てた１０年後の事業環境、システム環境、グローバル化を見据え、国内開発者はどう備えるべきかを考える。

過去から現在に至るまで、国内ＳＩは日本の経済成長とともに情報システムを担う仕組みとして、

①ハードウェア ： 自前で構築
②ソフトウェア ： 独自仕様を盛り込んだ手組みのソースコードによる
アプリケーション開発
③システム運用 ： 長期に運用に携わっている玄人によるシステム運用

といった痒い所に手が届くビジネスモデルが主流であったと考える。

今回、１０年後を考えるというテーマで運良く最新動向を得ることとなり、私自身が携わっているオフショア開発との関係性も合わせると、これからの国内ＳＩは、

①ハードウェア ： クラウドサービスによる利用するハードウェア
②ソフトウェア ： コーディングレス、オフショアによる低コストで統一化された
アプリケーション開発
③システム運用 ： アプライアンスの採用によるシステム運用の自走化・省力化

へ変化していくことが想定される。

この環境への備えとして、「残るものへの強化」、「進むものへのシフト、先行者利益」のアプローチが有効と考える。

「残るものへの強化」

環境の変化に左右されない領域を強化することで、システム開発の根幹となる専門領域を確立するアプローチ。情報システムの根幹となる「顧客レベルの業務知識」「システム全体をデザインする要件定義スキル」「インフラやアプリケーションを支えるシステム基盤スキル」は１０年後も常に求められるコアコンピタンスとなろう。

「進むものへのシフト、先行者利益」

変化する環境の先を読み、先行者利益を得ることで、ビジネスを獲得していくアプローチ。「海外リソースの活用ノウハウを熟知したなオフショア開発」「自動化アプライアンス導入実績・コンサルテーション」は、新たな需要を獲得するだけでなく、変化の先頭に立つものだけが習得できる未知のスキルを得ることもできよう。

６－６．１０年後の保守開発者への提言

最後に、保守開発者への３つの提言を行うことでまとめとする。
私自身も今後の１０年を環境の変化に流されない意思を持って取り組んでいきたい。

「一人ひとりの一芸強化を！」

冒頭で述べたように、若手の技術者が現場で幅広く開発スキルを習得する仕事が少なくなっており、たたき上げのＳＥが育つ環境が減っている。このことはジョブアサインにおいても同様に、技術者に何ができるかを詳細に問われる（入って覚える環境が無い）傾向が強くなっていると感じている。
一方で、お客様側がＳＥに対し将来的に何を目指すのかというキャリアパスを問うケースも増えている。
一人ひとりが得意とする領域・一芸を身に付けることが改めて重要であり、自信を持って仕事に取り組む環境を作るべきである。

「自動化のノウハウ・暗黙知の理解を！」

システム環境として、自動化・コーディングレスといった人手をかけない仕組みで海外リソースを活用した低コストの開発検討や導入が今後の１０年で大きく進んでいくと考えられる。
仕組みに乗るだけの仕事は海外リソースと競合するだけでなく、環境変化に依存する不安定な１０年後を迎えてしまう結果を招きかねない。
自動化の中にこそ今までのノウハウ・暗黙知が詰まっていると考え、仕組みのフレームワークや設計思想まで入り込んだ仕事の取り組みを行うべきである。

「海外リソースとの協働経験を！」

今後のシステム開発において、オフショア開発・海外リソースの活用が後戻りすることは無いと考える。
しかしながら、実際にオフショア開発を行っているケースはいまだ少ないと感じており、新規開発のケースでも経験の無さ、その不確実性から採用を見送る例も散見される。

かかる状況から、海外リソースと早期に協働して自ら体感することでノウハウを蓄積し、適用領域の判断を行えることが重要となる。
まずは自らの仕事にオフショア開発・海外リソースを少しでも取り込むべきである。

（岡田　浩）
（井瀬　英晶）

７．１０年後の保守担当に求められるスキル

はじめに

この１０年で、システム開発は着々と進化をし続けている。業務の自動化が飽和状態となっているため、お客様はより使いやすく戦略的なシステムを求め、価格競争が過熱化し個人情報の流出が社会問題化する状況下で開発はより効率的且つ高品質を求められる。

このような状況で、当然保守も変わっていかざるを得ない。１０年後、保守を担当する技術者がどのように変わっていくかを論述する。

７－１．１０年前のＳＥ

７－１－１．開発の体系

２０００年ごろ、Web システムが主流化してきた。
それまで、業務担当者が各自の PC にインストールしたり、TELNET 接続などを使ってコマンドベースで帳票を出したり売上を登録したりしていたのが、インストール不要で GUI ベースで操作できるシステムの登場により、Web システムへの乗り換えが各企業内で発生した。そのため当時のシステム開発は、VB で作ったアプリケーションや COBOL で作ったオフコンシステムのリプレイスがメインだった。そのため、元のソースコードをそのまま流用することはできないため、開発はスクラップ＆ビルドが主流だった。

当時からＯｒａｃｌｅなどが販売しているパッケージを使うことはあったが、既存システムができることは当然新システムでもできること　というのが暗黙のルールとなっていたので、パッケージのソースを直接いじる「カスタマイズ」もよくあった。

あるシステムをリプレイスすればその周りのインターフェースの開発も必要となり、次から次へと開発の依頼が舞い込み、ＳＥ＝帰れない　というのが常識となっていた。

当然そのような状況で、手順や整理は軽視されることが多く、開発手法はあっても形のみで、命名規則、設計書、データ構造は開発者任せであり、ひどい場合は開発言語も機能によって異なる場合もあった。また、設計書があっても初期開発のみでテストや途中の仕様変更が盛り込まれていないことは日常茶飯事であった。

７－１－２．保守担当者に求められたスキル

当時の保守担当者は、大抵機能単位（機能の粒度はシステムや会社によって異なるが）に割り当てられていた。

保守担当者の業務は、業務改定や組織変更などに伴うシステムの変更をエンドユーザーより依頼され、それに対してプログラム修正などを行っていた。
保守を受け持つ企業規模が大きい場合は、エンドユーザーとのやり取りを本社が請け、プログラム修正を下請け企業が行うこともあった。
そのため、保守担当者はエンドユーザーとのやり取りができるレベルの業務知識と、システムを修正することができるプログラムスキルが必要であった。
また、当時は開発の上で作成されるドキュメントがほとんど使い物にならなかったので、プログラムロジックから業務知識まで保守担当者は全てを把握していなければいけなかった。そのため、保守担当者は広範囲ではなく、機能単位など限定的な範囲で受け持つことが多かった。

７－２．現在のＳＥ

７－２－１．開発の体系

オフコンなどのリプレイスは一通り終り、企業が競って社内のシステムを入れ替えようという動きが薄くなってきた。
そのため、会社の統廃合などの特別な行事がない限りは、保守費削減や経営改善といった戦略的なシステムの導入が主流となってきた。
当然、このような状況であればシステムの発注も激減してくる。
そのため、価格競争も激しくなり、「いかに効率的にシステムを作るか」が重要になってきた。「つくらないものづくり」を推奨し、パッケージや部品の流用を義務化しているＩＴ企業も珍しくない。パッケージを使うときはテスト工数がかさんでしまう「カスタマイズ」は敬遠され、ＡＰＩなどを流用した「アドオン」「プラグイン」が主流となってきた。また、コーディングは単価の安い中国やインドへのオフショア化が日常化し、そのため、命名規則、ログ、データ構造、コーディングルールなどはコミュニケーションに不備が生じないようにガッチリとルール化及びメンテナンスされるようになった。

７－２－２．保守担当者に求められたスキル

設計書などドキュメントがガッチリとしてきた影響で、保守担当者がロジックや業務を全て頭に入れなければいけない状況ではなくなった。保守担当者はある程度の引継ぎ期間を与えられれば、設計書や業務マニュアルなどを参照しながらメンテナンスをすることができる。また、パッケージに関してはソースコードが読めることではなく、そのパッケージの機能の理解とパッチやリリース情報を理解できるだけの製品スキルがあればよくなった。そのため、「抱えていなければいけない人材」という状態がなくなってきたので、経費

削減の対象となり１人の担当者が見るシステムの範囲が広がり、エンドユーザーと話のできるレベルの業務知識の範囲は複数機能にまたいで必要になった。

７－３．１０年前から現在に至って

ドキュメント化がしっかりしてきたため、保守者は不明な箇所は設計書の索引であたりをつければよくなったため、細かいロジックを頭の中に叩き込む必要がなくなった。しかし、保守費削減のため広い範囲を浅く広く理解することを求められるようになった。
また、保守のオフショア化やパッケージを使った開発により、外部とのコミュニケーションが更に求められるようになった。

７－４．１０年後（２０２２年）の保守

現在、クラウド化が普及しつつある。
１０年後、各企業ではアプリケーションはもちろん、サーバーも自社でもつことが（サーバーを提供する企業は別として）まれになってくる。
サーバーも OS のインストールも不要で、申込のみで期間限定のお試し版システムを試行することができる。
また、業務が変化した場合は、現在使っているアプリの契約を別のアプリに乗り換えるだけで切り替えができる（ただし、データ移行は残る）。
市場には多くの企業が今まで各社で作っていた製品をクラウド化したものが出回り、そのため、多種多様なアプリケーションを試しながら選択することができる。
そのため、ＩＴ企業がエンドユーザーの御用聞きのような形で仕様を聞き、それをシステム化するといった１社：１社の形は激減し、ＩＴ企業の仕事は SaaS としてサービスするアプリケーションの開発か、エンドユーザーと一緒にマッチしたアプリケーションの選択支援といった形が主流となる。

７－４－１．開発の体系

クラウド向けのアプリケーションを開発する部隊と、エンドユーザーと一緒に戦略的に業務改革をするうえでのアプリケーション選び支援　といった２種類となる。
アプリケーションの乗換えをするときに必ず生じるインターフェース開発部隊も多少いるが、ほとんどが項目や媒体を設定ファイルに定義するだけの「開発」ではなく「設定」になるためアプリケーション選び支援が担う場合が多い。

開発は部品化が進み、また Web ではフレームワークはもとより、あらゆる機能が部品として無料もしくは有料提供されるので、システムあたりコーディングしなければいけないソースコードステップ数は現在の１割にも満たない。そのため、開発する部隊の割合は現在に比べて激減する。

７－４－２．１０年後保守担当者はどんな仕事をするのか

OS もサーバーも借り物なので、パッチあてなど基盤のメンテは不要になる。
システムも、会社向けに作られたものではなく、法改正などにより既存のシステムの改善が必要な場合は、部品を取り替えるか、もしくはシステムそのものを乗り換えてしまえばよいため、コーディングは不要。

ただし、新たなサービスを受ける、もしくは新たな部品を取り付けるための検証や調査は必要と成る。また、サービスの選択肢が膨大となるため、幅広い製品知識が必要になる。

７－５．１０年後（２０２２年）の予想

２０２２年にはシステムは「要望にあわせて開発する」のではなく「要望にあわせて取り替える」ものになる。

そのため、ユーザーからみるとシステムは「使い捨て」となる。
今まで、開発と保守の線引きは、システムの作り変えや大規模改造か、部分的な小さな修正かであった。

しかし、クラウド化により、作りこみによる開発がなくなるのであれば、開発と保守の線引きは不要である。

今後はクラウド向けのアプリケーションを開発する開発部隊　以外は全て「保守　兼アプリケーション選び支援」となり、市場のサービスに対する幅広い知識とお客様の業務知識の両方を求められる。現在人月２００万のコンサルタントと同等のスキルを求められ、保守担当は企業の将来を担う重要なポストとなるだろう。

（佐井　由美子）

８．おわりに

８－１．今年度の総括

私たちＣグループは、昨年のキックオフ合宿にて、「１０年後のソフトウェア保守を考えた時、誰が誰に何を伝承すべきなのか」をテーマとして取り上げた。はじめに、「１０年後のソフトウェア保守」の議論を行なった。１０年後は、クラウドサービスの普及や自律型サーバーへの移行が進み、「ソフトウェア保守環境は二極化する」と言う結論に至った。

二極化とは、企業における戦略的システムの保守は現行の延長で残り、戦略的システム以外のパッケージソフトなどは、今後クラウドを活用し、顧客の運用・保守がなくなるとことである。

まず、ビジネス競争に勝つための戦略的な業務については、ビジネスの進化に伴うシステム変更を行なう必要があることから、プライベートクラウドやレガシーシステムを企業が利用するようになる。プライベートクラウドやレガシーシステム環境におけるソフトウェア保守は、ビジネスとシステムを横断して行なわれるようになるため、より高度で幅広いＩＴ技術と業務知識が必要となる。古いソフトウェア保守については、古い技術や手法で保守が行なわれるが、新しいソフトウェアと繋がることにより、さらに複雑化する。

次に、ビジネスにおける汎用的な業務については、パブリッククラウドのサービスを多くの企業が共同で利用するようになる。パブリッククラウド環境では、システム運用・システム保守が自律型サーバーにより無くなり、自動化と無人化が進むことになる。アプリケーションは部品として提供されるようになり、部品の組み合わせや入れ替えでフトウェア保守が行なわれるようになる。アプリケーション自動作成ツールの進化により、ソフトウェア開発とソフトウェア保守の垣根はなくなっている。また、グローバル化（オフショア化）が進み、国境を超えたソフトウェア保守が行なわれるようになる。

何れのソフトウェア保守環境でも、スマートデバイスが活用されようになり、トラブル対応やユーザーサポートなどのソフトウェア保守業務の効率化に役立っている。

８－２．次年度に向けて

今年度は、「１０年後のソフトウェア保守」の議論に多くの時間を費やしたため、「誰が誰に何を伝承すべきなのか」については、あまり議論が進まなかった。若手エンジニアの育成は、新旧のソフトウェアが混在するソフトウェア保守環境において、今後は重要度を増していくと思われる。

次年度については、今年度あまり議論が進まなかった「誰が誰に何を伝承すべきなのか」を継続して活動し、伝承するための方法論やツールなどの研究を進めて行きたい。

最後に、フォーラムのゲストスピーカー引き受けていただいた方々、並びにフォーラムにて積極的に議論に参加していただいた方々に、この場をお借りして御礼を申し上げさせていただきます。

以　上

第２１年次 SERC 個別テーマ

「ＳＥＲＣの考える保守とは」
活動報告

メンバ

「仕事品質」改善教室	大島　道夫
（株）日立ソリューションズ	鈴木　勝彦
（株）日立ソリューションズ	高橋　宏志
（株）日立ソリューションズ	高橋　芳広
NARA コンサルティング	奈良　隆正
東芝ソリューション（株）	野口　大輔
（株）精密形状処理研究所	長谷川　亨
（株）NTT データ CCS	馬場　辰男
（株）バイトルヒクマ	弘中　茂樹
東芝ソリューション（株）	増井　和也
（株）日立ソリューションズ	松本　道春

目　次

０．活動概要

１．改めて「ソフトウェア保守」とは
１．１　ソフトウェア保守のスコープ・用語一覧
１．２　メンバーの考えるソフトウェア保守
１．２．１　開発と保守について（鈴木勝彦）
１．２．２　保守とはソフトウェアへの愛情をもったケアである（増井和也）
１．２．３　ソフトウェア保守の地位向上を願って（野口　大輔）
１．２．４　保守業務改革のための「保守プロセス整備とマイビジョンの設定」のすすめ
（大島　道夫）

２．ＳＷＥＢＯＫ　Ｖ３（ＤＲＡＦＴ版）　第５章　ソフトウェア保守

３．啓蒙・広報活動
３．１　SERC フォーラム：「大規模障害に学ぶソフトウェア保守」開催

０．活動概要

作業部会活動記録

下記は，第２１年次Ｄグループ作業部会会議および研究活動の経緯を時系列に示したものである。

● ソフトウェア・メインテナンス・シンポジウム　２０１１

日程：　2011 年 10 月 14 日（金）

場所：　東銀座 JJK 会館

テーマ :『ソフトウェア保守の新たな道』

議題：　20 年次活動報告，21 年次活動計画の発表

● 第１回定例会

日程：　2011 年 12 月 9 日（金）~10 日（土）

場所：　三島商工会議所会館

議題：　ノウハウ集作成手順

● 第２回定例会

日程：　2012 年 1 月 26 日（木）

場所：　日立ソリューションズ

議題：　ノウハウ集作成手順

SERC フォーラム検討

● 第３回定例会

日程：　2012 年 2 月 17 日（金）

場所：　NARA コンサルティング

議題：　SWEBOK　V3

SERC フォーラム検討

● 第４回定例会

日程：　2012 年 3 月 16 日（金）

場所：　NTT データ CCS

議題：　みずほ障害報告書

SERC フォーラム検討

● 第５回定例会

日程：　2012 年 4 月 20 日（金）

場所：　日立ソリューションズ

議題： SERC フォーラムの準備

● SERC フォーラム

日程： 2012 年 5 月 18 日（金）

場所： 八丁堀区民館

タイトル：大規模障害に学ぶソフトウェア保守

～ソフトウェア保守者が見た　みずほ銀行オンライン障害～

● 第 6 回定例会

日程： 2012 年 6 月 16 日（金）

場所： 東芝ソリューション

議題： SERC フォーラムの反省

私の考えるソフトウェア保守

● 第 7 回定例会

日程： 2012 年 7 月 19 日（木）

場所： バイトルヒクマ

議題： 私の考えるソフトウェア保守

● 第 8 回定例会

日程： 2012 年 8 月 31 日（金）~9 月 1 日（土）

場所： 朝日の宿「真鶴荘」研修室

議題： 活動報告書作成

（文責：高橋芳）

１．改めて「ソフトウェア保守」とは

1.1　ソフトウェア保守の用語・範囲一覧

ソフトウェア作業の現場では，ソフトウェア保守を指し示す用語や，その名前で実行される作業の範囲は様々である。ソフトウェア保守のプロセス，技法，ツール等を議論する場合に，それが指す物が一致しないと，議論が成り立たなくなることがある。そこで，ソフトウェア業界で保守を示す言葉を収集し一覧を作成した。ソフトウェア保守の議論の一助となれば幸いである

名称	定義元等	作業（○：含む，　－：含まない）						
		ﾄﾗﾌﾞﾙｼｭｰﾄ	不良訂正	ﾄﾞｷｭﾒﾝﾄ訂正	性能向上	ﾌﾟﾗｯﾄﾌｫｰﾑ拡張／確認のみ	ﾌﾟﾗｯﾄﾌｫｰﾑ拡張／修正有	機能追加
ソフトウェア保守	ISO/IEC 14764, JIS　X0161	○	○	○	○	○	○	○
	（増井）	○	○	○	○	○	○	○
ソフトウェア保守開発	SERC	○	○	○	○	○	○	○
保守開発	データ総研	○	○	○	○	○	○	○
フィールドサポート	一般	○	－	－	－	－	－	－
	（鈴木）	○	○	○	○	○	－	－
エンハンス	野村総合研究所	○	○	○	○	○	○	○
	（高橋芳）	－	－	－	○	－	○	○
派生開発	派生開発協議会	－	－	－	○	－	○	○
ソフトウェアセービング	（野口）	○	○	○	○	○	○	○

※括弧はメンバー採取情報

（文責　高橋芳）

1.2　メンバーの考える「ソフトウェア保守」

1.2.1 開発と保守について（鈴木勝彦）

（１）開発とは(パッケージ製品の場合)

・新規開発と機能追加

(a)Ver1 が新規開発

Ver1 開発時に流用があってもよい

(b)Ver1 以外の Ver1.1 や Ver2.0 などが機能追加

不良吸収のみは機能追加と言わない

(c)個別対応でも機能追加があれば開発

（２）保守とは(パッケージ製品の場合)

・新規開発/機能追加以外で，製品リリース後に発生したあらゆるものに対応すること。

(a)社外事故対策版の作成

(b)品質向上で見つけた不良吸収版の作成

(c)社外事故調査

(d)問い合わせ対応

(e)マニュアル訂正

(f)修正を伴わない動作確認(新 OS など)

（３）開発と保守との差(パッケージ製品の場合)

・新規開発/エンハンス以外で，製品リリース後に発生したあらゆるものに対応すること。

(a)保守は，機能仕様書までの上流工程がない

(ニーズの把握や機能/仕様を決めることがない)

(b)保守は，顧客対応や現地対応することがある

(現地調査や謝罪)

(c)開発は，製品説明会などがある

(d)担当によっては好き嫌いがある

(必要なスキルはあまり変わらない)

(e)保守は，予定が立たない(突然徹夜になる)

(f)「保守」という言葉の響きがよくないので，職場では「フィールド/サポート」と呼んでいる。

（文責：鈴木）

1.2.2 保守とはソフトウェアへの愛情をもったケアである（増井和也）

（1） ソフトウェアライフサイクルについて

●SLCP 国際規格の大改定‥「開発」が消えた？

SLCP（Software Life-Cycle Process）の国際標準規格として ISO/IEC 12207（初版 1995 年，第 2 版発行 2008 年。以下第 2 版を SLCP 規格，初版を旧 SLCP 規格と呼ぶ）がある[1]。SLCP 規格は日本では JIS 規格[2]になっている。また，旧 SLCP 規格を基本ベースに，日本の事情を加味し，日本におけるソフトウェアの共通フレームとして何度かの改訂が行われ，現在『共通フレーム 2007　第 2 版』[3]として刊行がなされている。

筆者が注目する SLCP 規格の旧 SLCP 規格からの大きな変更点は，ソフトウェアに関し「開発（development）」及び「開発プロセス」という用語を極力使わず，「実装（implement）」及び「実装プロセス」という用語に変わっている。「開発」の代わりに「実装」を使用することで「開発は新規性豊かで想像的作業」という特別な「開発」イメージが出ることを排除しようとしたのだろうと筆者は評価している。なお，「開発者（developer）」という言葉が依然最新の SLCP 規格で使われている部分があるが，同規格の「用語の定義」では「実装者（implementer）と同義」とされている（SLCP 規格 4.10 及び 4.15）。

SLCP 規格上，今まで慣れ親しんだソフトウェアの「開発」という概念が事実上無くなったことになる。いっぽう，「保守」，「運用」の各プロセスは，多少の説明変更はあるが「テクニカルプロセス」という「実装」プロセスも含まれる上位範疇の中に入り，従来通りの名称で残っている。なお，旧 SLCP 規格の保守プロセスにおける最後のアクティビティであった「廃棄（retirement）アクティビティ」は「廃棄プロセス」に昇格し，保守プロセスと分離して記述されている。

いずれにしても「そのソフトウェア作業は開発か保守か？」とか「保守は開発の一部であるか？」といった議論は最新の SLCP 規格に準じる限り，無意味ということになる。筆者が「『開発』という言葉の無闇な拡大適用がソフトウェアの適正なプロセス概念を乱している」という長年の主張がようやく SLCP の国際規格においても結果として実現された形である。本稿（1.2.2）でも SLCP 規格に準じ「開発」は原則使わない。

●心理学的なライフサイクルとの関係

さて，SLCP 規格で使われている「ライフサイクル」という言葉は，エリク．H．エリクソン（1902～1994）という発達心理学者が 1959 年頃最初に使ったとされている[4]。エリクソンは「アイデンティティ」という一般にもなじみのある心理学用語を最初に定義したこと[4]でも有名である。

エリクソンはヒトが生まれて一生を終えるまでの生涯（ライフサイクル）では次の図 1.2.2-1 のように 8 段階の状態を経ると書いている[5]。

① 乳児期（０歳～１歳半ごろ）‥基本的信頼　対　不信感
② 幼児期前期（１歳半～３歳ごろ）‥自律性　対　恥・疑惑
③ 児期後期（３歳～６歳ごろ）‥積極性　対　罪悪感
④ 学童期（６～１３歳ごろ）‥勤勉性　対　劣等感
⑤ 青年期（１３歳～２２歳ごろ）‥同一性（identity）　対　同一性拡散
⑥ 成人期初期（２２歳～４０歳ごろ）‥親密性　対　孤立
⑦ 成人期（壮年）‥生殖性　対　自己停滞
⑧ 老年期（高齢）‥統合性　対　絶望

（注）「○○　対　××」は「対」の左辺と右辺と葛藤状態（これを発達課題と呼ぶ）のもとで，両方を経験する時期であるとエリクソンは説明している。

図 1.2.2-1　E.H.エリクソンのライフサイクルと発達課題

なお，心理学用語以外では，経営学において製品のライフサイクルマネジメント（PLM）に関する研究や実践がなされている[6]。

SLCP 規格での「ライフサイクル」の定義は「システム，製品，サービス，プロジェクト，他の人工物の概念から廃止までの漸進的な変化」（SLCP 規格 4.16）となっている。一見，定義は PLM の定義に近いようだが，最後の「漸進的な変化」とある点で，エリクソンの心理学的概念の「ライフサイクル」も想定しているのではないかと筆者は考える。また，SLCP 規格で development（開発）を使わず implement（実装）を使うように変えたのは，エリクソンのヒトの「発達」は英語では「開発」と同じ development のため，混同を避けたためではないかと筆者は推測する。

●ライフサイクルと保守

SLCP 規格の保守プロセスの「修正の実施アクティビティ」では「テクニカルプロセスを実行して修正を行う」とある（SLCP 規格 6.4.10.3.3.2）。「テクニカルプロセス」には「実装プロセス」（SLCP 規格 7.1.1）があり，そこでソフトウェアの修正（実装）が行われると見ることができる（旧 SLCP 規格でも保守プロセスから開発プロセスが呼ばれるとある）。この点で，SLCP 規格では稼働後のソフトウェアでは，保守プロセスが実装プロセスより上流にある（保守プロセスが先に実行され，そこから呼ばれた実装プロセスの結果を受け入れる）と筆者には見える。

保守プロセスは，SLCP 規格で定義されたテクニカルプロセスに分類される 11 のプロセスの中で，唯一より詳細な国際標準規格が作成されている。それは，1999 年初版が出され，2006 年に改訂版が出されたソフトウェア保守プロセスの国際標準規格 ISO/IEC4764 IEEE Std 14764[7]である（JIS 規格は X 0161-2008[8]）。SLCP 規格の下位規格の位置ではあるが，保守プロセスのみ別に特化した詳細国際規格を作成した事実から，ISO（国際標準化機構），IEC（国際電気標準会議），IEEE（The Institute of Electrical and Electronics Engineers, Inc.）が SLCP の中で保守プロセスをもっとも重要視している証しだろうと筆者は考えている[9]。

（2）保守へはいつ変わるのか？

●保守の準備は生まれる前から，そして保守の開始は社会人から

ソフトウェアの新規実装は，エリクソンのライフサイクルの８段階のどこで終り，保守に引き継がれるのかを考えてみる。筆者は以前ヒトが生まれるまで（母親のお腹の中にいる妊娠期）が新規開発で，生まれてからは保守に引き継がれるのが分かりやすいと考えていた。

しかし，今年度の作業部会の議論の中で，ソフトウェアのライフサイクルの場合，それでは説明できない場合があることを感じた。新規実装中のソフトウェアはまったく利益をもたらさない（実装費用のみ発生)。稼働して初めてソフトウェアの目的が達成され，その効果によって利益が導かれる。

ヒトの場合，本人のみの意思で最初に所得を得るタイミングがそれと考えれば，生まれてすぐではなく，学校を卒業し，仕事についたときを稼働と考える方が説明しやすいだろう。

そのように考えると次の図 1.2.2-2 で示す⑤青年期が済んだ段階で新規実装から保守に引き継がれるのが妥当といえる。

［妊娠期］‥構想～要件定義

① 乳児期‥概要設計中（レビューによる修正含む）⇒要件の信頼感の共有

② 幼児期前期‥基本設計中（レビューによる修正含む）⇒自律的設計内容の共有

③ 児期後期‥詳細設計（レビューによる修正含む）⇒積極的な高度設計の取込み共有

④ 学童期‥製造，単体テスト，結合テスト（テストによる修正含む）⇒集中した（勤勉な）製造，テスト実施の共有

⑤ 青年期‥システムテスト，運用テスト（テストによる修正含む）⇒対象ソフトウェアのアイデンティティの共有

＜＜⑥からが保守。ただし，SLCP 規格では①～⑤の段階で保守プロセスの「プロセス実装」アクティビティとして保守者が先行参画することを推奨している＞＞

⑥ 成人期初期‥保守プロセスへの引き継ぎ，是正保守中心⇒稼働ソフトウェアがユーザに受け入れられる過程（親密性の過程）の共有

⑦ 成人期‥予防保守，適応保守中心⇒環境変化による新たなニーズに対する機能強化（子孫を増やす）の共有

⑧ 老年期‥完全化保守中心⇒保守を繰り返すことによって発生する不整合部分の総合の共有

［死亡］‥廃棄

（注）なお，次の各段階の説明中「⇒」で始まる説明部分は，エリクソン発達段階の説明を意識して書いた関係者（取得者，供給者，各プロセス実施者）にとっての課題である。

図 1.2.2-2　エリクソンのライフサイクルをソフトウェアへ適用

●幼過ぎる内に働かせると後が大変？

エリクソンのライフサイクルは，第二次世界大戦後の先進国（アメリカ）の一般市民を対象としているように見える。しかし，全世界では億人単位の児童が大人と同じ過酷な労働をさせられ，街頭や路上でモノを売らされ，中にはそのわずかな収入で親を扶養，また親とも離れ離れとなり自らがストリートチルドレンとしての生活を余儀なくされている姿を想像させる統計がある（2000 年 ILO 統計）。

ソフトウェアの新規実装ではどうだろうか？　十分に育てず（必要機能の網羅性を高めず），その上でテストを十分繰り返さずに稼働をさせ，結果として幼い学童が大人と同様の成果をもとめられるような状況を招き，稼働後多くの問題を発生させる新規実装ソフトウェアがなくならない。ソフトウェアの新規実装における発達段階の見極め（稼働判定や稼働までのコストの掛け方）について，エリクソンのライフサイクル（ヒトの生涯）に学ぶ点は多いと筆者は考える。

（３）　ソフトウェア保守に必要なもの何か？

●稼働前と稼働後の違い

ソフトウェアの新規実装プロジェクトにおける実装や修正の作業と，既存ソフトウェアの保守段階での修正作業との大きな違いは，稼働前か稼働後の違いである。稼働前ではテストなどで発見された修正要件対応は，一般にプロジェクト計画に想定期間を設けることが可能である。仮に新規実装プロジェクトのテストで修正要の部分を修正する余裕も無く，計画外の労働量をメンバーの依頼や予定外の要員を大量に投入しなければならなくなった場合，その実装プロジェクトはプロジェクト計画通りにできなかった（またはプロジェクト計画自体がずさんだった）失敗プロジェクトと評価されるだろう。

稼働後の保守案件対応では，修正スケジュールのコントロールは稼働システムへの影響や，放置による機会損失額がどの程度かで緊急性が決定され，保守者側の都合だけで計画を立てられないケースが多い。また，新規実装プロジェクトのように綿密なプロジェクト計画立案及び計画レビューに多くの時間と強力な組織的パワーを掛ける余裕がないことも珍しくない。

●愛情を持ったケアが保守のポイント

このような保守案件対応で「新規実装にはない対応要件は何か？」を SLCP 規格が示す保守プロセスを念頭にポイント 9 項目を次の表 1.2.2-1 に示す。基本は既存ソフトウェアをヒト（生き物）と考え，愛情や倫理観を持って接する（ケアする）気持ちが大切と筆者は考えている。その理解を助けるため，医療に当てはめた例を示す。

表 1.2.2-1 ソフトウェア保守，ヒトの医療，ソフトウェアの新規実装

No.	保守に必要なポイント	医療で喩えると	新規実装との考慮点相違
1	保守対象システムやソフトウェアに対する詳細で十分な事前の理解	医師が人間の身体の仕組みを十分知っているように	段階的詳細化により，順次実装方式を明らかにしていく。そのため，最初から最終的な仕組みや詳細を知っている必要はない。
2	保守案件の現象の正確な理解	医師が患者の症状を正確に把握するように	新規実装のテスト段階で発生した修正要件対応は，一般に実装者自らが現象確認するため，再現させやすい。
3	対象システムやソフトウェアの運用実態や更新の経緯に対する正確な理解	医者が患者既治療やどんな生活をしてきたかを正確に理解しようとするように	新規実装では考慮の必要が少ない。
4	ユーザに対して最適な修正方法や運用回避方法を提示・説明する技術	医者が傷病の原因について丁寧に説明し，治療方法や症状悪化の進行を遅らせる方法を患者に説明して，納得を得るように	新規実装時には，プロジェクト内で多くが処理され，ユーザやステアリング委員会に説明する内容は一部に限られる。
5	最適な修正を素早く正確に実施する技術	医者が適切な投薬処方箋を書く，外科医に手術を正確に依頼する技術を持っているように。または自らが施術するように	新規実装では，テストで発見された修正案件は実装者がある程度まとめて対応が可能。
6	修正の実施結果に対し正当な評価・レビューができる技術	医師が投薬や手術の結果を診て，傷病の治療の効果を確かめるように	新規実装では，修正結果の確認はテスト担当者や品質保証部が最終的に確認するなど組織的役割分担が明確。
7	修正が適切に完了し，リリースしても良いと判断できる技術	医師が治療終了や完治を患者に宣言するように	新規実装では，品質保証部やステアリング委員会がまとめて稼働判断を最終的に行う。
8	保守案件対応の管理スキル	医師がカルテを法定 5 年以上保存しておくように	新規実装では実装者がテストで発見された問題一覧を納品しない。また，多くの担当者は記録に残さない。
9	システムの定期診断の提案	医師が積極的に定期的な健診や受診を勧めるように	新規実装では，稼働の瑕疵担保期間は瑕疵が発生するまで待ちの状態。

（4） ソフトウェア保守の今後の姿について

●より高度な保守者，保守プロセスが必要

既存ソフトウェアの量が増え続けていくなら，既存ソフトウェアの量単位に必要な保守者数もより多く必要（正の相関がある）と筆者は考えている（人口が増えればそれに応じた医師の増加が必要なように)。それを証明するかのように，情報システム部門を持つユーザ法人は，システム

維持費（保守・運用費）が将来増加の一途をたどることが経営のコスト構造を硬直化させるリスクを負うことになると考え，パッケージソフトウェア，ASP（Application Service Provider），SaaS（Software as a Service），シェアドサービス，クラウドサービスなどへの移行を行い，保守・運用すべき既存ソフトウェアの量を減らす努力をしている。今の日本では医療費や教育費の家計負担が大きいため，親が多くの子供を持たないようにである。

では，最終的にそういったパッケージソフトウェアの導入や各種アウトソーシングサービスをユーザ企業のすべてが採用すれば，保守者は不要になっていくのか？　答えは「ノーである」というのが筆者の予測である。既存ソフトウェアは，パッケージベンダやASP，SaaS，シェアドサービス，クラウドサービス等のプロバイダ，各種製造業（組込みソフトウェア）などとして存在し続ける（既存ソフトウェアの絶対量は各ユーザ法人で所持するより減るかもしれないが）。当然，そのソフトウェアの保守者が必要となる。

また，IT サービス等の利用法人は競合企業との差別化を図るため，最終顧客に提供する IT ベースのサービスを競合他社よりも優れたものにする必要があるすれば，他社にない独自のサービスを提供するソフトウェアの個別所有（実態はサービス提供業者内にあったとしても）を戦略的に行う必要が発生する。そういった戦略的ソフトウェアが不要なら，最終顧客に対する IT サービスが企業戦略上の重要成功要因（CSF）ではない（他社と横並びで良い）ことになってしまうが，そんなことがあり得るだろうか。

国内・国外と熾烈な競争を行っている法人にとって最終顧客に対する IT サービスが CSF で無くなることは考えにくいとすれば，そういった独自ソフトウェアの保守者はやはり別に必要となるのである。

大局的には価値が高いソフトウェア（※）は長く残り，価値の低いソフトウェアは価値の高いソフトウェアにとって代わられていく（淘汰されていく）。

※ 高いシェアを持つ OS（Operating System）／MW（MiddleWare）／各種アプリケーションのパッケージソフトウェア，ASP・SaaS・シェアドサービス・クラウドサービスなどのプロバイダが所有するソフトウェア，各種製造業の組込みソフトウェア，企業が独自に所有する戦略的ソフトウェア等

結局，ソフトウェアの価値が高ければ高いほど，最適な保守プロセスとそのソフトウェアに対してより優秀な保守者が一定量必要となる。その結果，より高度な保守技術や保守プロセスが成熟していくと筆者は予想する（世界をリードする国に，高度な医療技術が集中・確立していくように）。

筆者はソフトウェア技術を語る際，従来の「新規開発まずありき」という従来の発想をすぐにでも変える必要があることをこれまで以上に強く訴えたい。新規開発オリエンテッドな考えに固執すると，たとえ日本の研究所やソフトウェア業が世界をリードするような戦略的ソフトウェアを多額の研究開発投資を使い，開発できたとしても，それで終わってしまう可能性が高いからである。せっかく開発したものも，育てられない，競争力を維持できない，シェアを拡大できない，儲けられない，投資を回収できない，といったことの繰り返しになるのではないかと筆者は考える。「モノができれば後は何とかなる」というライフサイクル全体の最適化をほとんど考えない視状況を打破するための施策を提言として述べ，本稿とまとめとする。

●筆者の提言

外国の有力なパッケージソフトウェアベンダーやインターネットサービスプロバイダ（例：海外の M 社，A 社，O 社，G 社，Y 社，F 社）を買収するを試みる。そして，買収した企業の所有ソフトウェアを評価し，日本人の得意とするきめ細かい改造を施し，特定分野で競争優位なものにしていく。そのことが将来可能となるような既存ソフトウェアへの付加価値技術の研究開発を今から行う（各社の各部門に内在している当該技術を集約して)。

日本人は今までにない独創的なものを造るより，既存のモノに改造，改良を繰り返し，他への応用・転用も積極的に行い，ニッチな要求も満足させ，やがて大きな市場に育て，競争力の高いものにしていくことは得意な分野だと思われる。ソフトウェアも同じではないだろうか。

参考文献

[1]ISO/IEC 12207:2008 Systems and software engineering -- Software life cycle processes

[2] JIS X 0160:2012 ソフトウェアライフサイクルプロセス　日本規格協会

[3] 共通フレーム 2007 第 2 版 ～経営者，業務部門が参画するシステム開発および取引のために～ 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

[4]E.H.Erikson "Identeity and the Life Cycle",1959. Psychological Issues Vol 1,No.1.Monoguraph1., International Universities Press

[5]E.H.Erikson “The Life Cycle Completed” ,1982.　W・W・Norton & Cpmpany

[6]John Stark　”Product Lifecycle Management: 21st Century Paradigm for Product Realisation”,2011. Springer-Verlag

[7]ISO/IEC 14764:2006 IEEE std 14764:2006
Software Engineering — Software Life Cycle Processes — Maintenance

[8]JIS X 0161:2008　ソフトウェア技術—ソフトウェアライフサイクルプロセス—保守　日本規格協会

[9]増井，弘中，馬場，松永　～ISO14764 による～ソフトウェア保守開発　ソフトウェア・メインテナンス研究会編　2007　ソフト・リサーチ・センター刊

（文責：増井）

1.2.3 ソフトウェア保守の地位向上を願って（野口　大輔）

（1） 保守者に代わる名称案

「ソフトウェア　セーバー」(Software Savior)

直訳するとソフトウェアの救助者・救済者。(ライフセーバーをもじって）つまり，ソフトウェアの障害や不良を修復する人。(ちなみに保守は「ソフトウェア　セービング」となる）

緊急保守の対応にイメージが近いが，ライフセーバーが溺れかけた人を救う（海難事故の救助）活動だけでなく，海浜の事故防止そのものを防止することも任務としてある（ソフトウェア保守でいうと「予防保守」に近いか)。ソフトウェア保守の定義には「緊急保守」「適応保守」「予防保守」「完全化保守」があるが，現状一般的に保守のイメージは「緊急保守」だと思われるので，あえてイメージが良い（響きが良い）言葉で，かつ一般者にも理解しやすいと思われるので，この名称を提案する。

（2） なぜ名称にこだわるか

ソフトウェアは今後ますます増え，それを保守メンテナンスする仕事は増え，重要性は増していくと思われるが，正直，保守は新規開発に比べて　低く見られている。特にソフトウェアの開発や保守に直接携わったことがない，現場を知らない人の中には保守不要論者もいる。新規開発時にバグを無くせば保守は不要と考えているようだが，ソフトウェアと全く関係ない人が言うなら兎も角，影響力のある人がそのような考え方をすると，ますます保守の「地位」が低下してしまい，現場で激務に耐えながらも，一生懸命に仕事する人が不幸になるばかりである。それを何とか改善していくためには，まず，職業としての「保守」の地位を向上させる必要があり，まず「イメージ」を良くすることはやはり重要であり，あえて名称にこだわってみた。

（3） 保守の例え

医療のメタファーには全く及ばないが，医療以外に例えがないか考えてみたところ，野球にも例えられるのではないか（スポーツであるが，職業にしている人もいるという意味で）

新規開発が先発ピッチャーで，保守はリリーフピッチャー。昔は先発完投型が主流だったが，最近は，ほとんど抑えのピッチャー（リリーフピッチャー）にバトンタッチする。相手を抑えるとセーブポイントがつく。(「セーバー」にも近い）ちなみに相手とは，トラブルであったり，競合他社かもしれない。

（4） 保守の重要性

今後世の中（特に「先進国」）は，少子化に伴う人口の減少やデフレなどにより，経済成長率は益々低下していくと予想している。コンピュータシステムも例外ではなく，新規投資を控え，現状のものを改良しながら使い続ける動きが主流になっていくと思われる。投資する分野は，①効率化（人員削減，実作業者の不足にともなう）　②社会福祉　③新規エネルギー　④発展途上国が中心となると思われるが，①の効率化に対する投資も，全面的にシステムを入れ替えるのでは

なく，極力既存アプリケーションを流用することになっていくであろう。したがって，保守の重要性は益々高まるものと思われる。

（５） 保守資格について

保守者は単なるバグ修正を行うだけではなく，機能強化，さらにはお客様の運用を含めた改善案の策定，提案なども求められると思われる。このように，保守者にも各自が持っている技術レベルをステップアップさせ，それを証明するための資格制度（ソフトウェア　セービング資格）を設けることを提言する。例えば，３段階程度のレベルを設け，講習会とテストを行い，合格すれば「ソフトウェア　セーバー」として認定されるようなものがあると良いのではないか。１級～３級まであり（名称は何でも良いが），３級は「ソフトウェアの保守概念を理解し，ソフトウェアの修正ができる」。２級は「さらに，ソフトウェア保守理論を理解し，ソフトウェアの修正（緊急保守）だけでなく，適応保守や予防保守も行うことができる」，１級は「ソフトウェアを中心に，お客様のシステムの根本的な問題を追求し，運用含めた課題解決し，顧客満足度を向上させることができる」など。このような資格制度があり，それが世の中で認められるようなものになれば，保守者のモチベーションも上がり，保守のイメージ向上にも貢献し，保守者を目指す者も増えるのではないか。

（６）最後に

セーバー「Savior」のもう一つの意味

「救世主」

つまり，「イエス・キリスト」のことも指す。

諸外国ではお客様と「契約」を結ぶことで，契約書に記載されていることを履行することが目的で，お客様とは比較的対等な立場であるのに対し，日本では，古くから「お客様は神様である」という習慣があり，極端に言えばお客様の言われることが絶対である，とされてきた。今後は保守者も「神様」になるとまでは言わないが，お客様含め世の中に認められるようになり，自尊心を持って，対等な立場になれると良いと思う。

保守に携わる人々が高い志を持って臨める職業になることを願って。

以上

（文責：野口）

1.2.4　保守業務改革のための「保守プロセス整備とマイビジョンの設定」のすすめ（大島　道夫）

「IT 企業※1 の保守業務に従事している方が，自らの仕事に自信を持ち，品質向上，生産性向上そして売上/利益拡大を実現し，評価されることで IT 企業がもっと元気になれないものか。」「そして，この実現に何らかの形で貢献したい。」

これが IT 企業で約３０年間勤務してきた私の課題認識です。
現在，IT 企業の社員数の 6～7 割りの方が保守業務に従事していると言われています。
また保守業務に従事されている多くの方は，更なるお客様満足度向上を目指して，品質向上，生産性向上等に日夜努力され，その成果をあげています。
しかし，一方で保守業務に従事している一部の方が，悩みを抱えておられることも事実です。その悩みとは，

「出来たら開発部門等の評価されやすい部門に異動したい。」
「しかし，オンサイト勤務が長期化し，同じ職場/システム環境であったため，
オンサイトの固有技術しかなく，また新たな職場での人間関係にも不安がある。」

IT 企業の保守業務部門に配属された当初の理由は，人それぞれでしょうが，ある時期になると同様の悩みを持たれるようです。そして，その悩みは日々の仕事に忙殺され，いつの間にか「まあ，いいか」に変わってしまうようです。
また会社では，毎年，新しい戦略や戦術が企画され，各部門に示達されますが，保守業務部門については，旧態依然のムードが漂い，なかなか改善が進みません。

これらのことは，私が約３０年間勤務した IT 企業でも同様でした。現在の職場や評価に不満があって仕事をしているようでは，成果物品質はもとより，個人の精神面にも良くはありません。ましてや一部の方の悩みとは言え，IT 企業の 6～7 割りの方が保守業務に従事していると言うことは，経営問題にもつながっているかも知れません。
逆を言えば「IT 企業の保守業務部門に従事する社員の活性化こそ，業容拡大のキーワード」とも言えるわけです。

恐らく，IT 企業は，業容拡大のために要員配置の方法を長きにわたり優先してきたと思われます。当初は，配置した要員管理中心でもビジネスになっていました。しかし，お客様ニーズが「安定した要員配置」から「システム管理運営やシステム改善への提案要求」に変化したことに気がつかず，また気がついても，その対策が遅れたことにより，旧態依然の職場風土となってしまったのではないかと思われます。

またスキルアップ計画については，自社内でシステム環境が無いために，お客様の OJT に頼らざるを得ないという勘違いをしていたこと。さらに技術者本人も現業をこなすのに精一杯で勉強する余裕が無かったことなどがあげられます。

今，多くの IT 企業が現状の打開策を急がれています。

しかし，何故，このような状況になったかの個別の原因をこれ以上追及しても，全て過去のことで有り，あまり意味が無く，その時間もありません。また原因がわかったとしても本人や会社が「あの時は，こうすべきだった」と振り返りをしても殆ど解決できません。それより「これからどうするのか」の視点で考える方が，ワクワクするでしょう。

現状の課題にどのように取り組むかは，極端に言えば，「やり方」と「やる気」が大きく関わってきます。つまり，以下の式が成り立ちそうです。

「仕事の成果」=「やり方」×「やる気」

「やり方」とは，プロセス改善のことです。これは，会社つまり組織が主導で行う活動です。特に負のスパイラルの打開策として取り組むプロセス改善は，必要な体制とコストを組織として本気で投資する必要があります。現場レベルのプロセス改善でも一定の効果がありますが，負のスパイラルの打開策として体制とコストをかけないでプロセス改善に取り組むのは，中途半端になって結局上手くいきません。

プロセス改善は，まず保守業務の標準プロセスを組織として定義し，標準プロセスをテーラリングして，個別のプロジェクトに最適なプロセスとして計画し，実行します。

また決められた活動を測定しながらデータを蓄積します。プロジェクト完了後に測定されたデータや観察事項等をもとに改善項目を振り返ります。

改善が必要な場合は，標準プロセスを変更します。また社員がこれらのことを実行できるように組織をあげて継続的に教育を行います。そして，これを継続的に繰り返すことにより，保守業務プロセスの属人化を防ぎます。

保守業務では，ある案件の見積もりや影響度調査等の工数やテスト等の工数が多くなる傾向があり，このプロセスにも属人化の課題があります。この課題解決には，プロセス改善の一環としてドキュメントの最新化，共有化等などがあげられます。またツール導入による対応策も有効です。

本人が「やる気」を出すためには，マイビジョンを設定し，自ら成長したい目的を明確にすることです。目的を明確にすることで「そのためには，〇〇をしなければならない」ということが自ずと導きだれます。

目的を明確にしていない状態で，上司に「〇〇しなさい」と言われても，やりたくない理由を探してしまいます。特にオンサイト先での勤務が長くなると「自分の会社は，何もしてくれない」，「こんな職場環境だから，しょうがない」等と「依存型思考」になってしまう場合があります。このような「依存型思考」の状態の時には，周りからの細かい指示は，なかなか受け入れて

もらえません。

まずマイビジョンを設定し，「そのために自ら○○する」と周りの方にコミットメントする事が必要です。

「依存型思考」を脱するためには，自分が変わることでしか出来ません。そのためにもマイビジョンを設定し，周りの方から見てもらい，客観的な視点からの修正を繰り返すことで自分の考え方の過不足に気が付き，次第に腹落としが出来る様になります。

自分の「やる気」が明確になると共に，他社でも通用する保守業務プロセス，「やり方」を習得することで「自分の仕事に自信がもてる」様になります。

これまで述べてきた現状と期待値にその原因と問題点等を仮説として加えて，別紙-1「表-1 IT企業の保守の現状と対策案」に整理致します。

※ -1
IT企業:システムユーザーまたはメーカー等の下請けで，情報サービスを生業とした企業をいう。メーカー系や独立系を問わない。自社パッケージ開発/販売系の保守，システムユーザー等の情報システム部門等にも該当するかも知れないが現段階では除く。

表-1　IT 企業の保守の現状と対策案〈仮説〉

区分	現状	期待値	問題点	原因	対策案
IT 企業 (組織)	①単価/売上/利益共に頭打ち。 ②社員教育ままならず。 ③長期配置による社員のモチベーション低下	①右肩上がりの業績確保 ②継続的な研修の提供による社員スキルアップ/満足度向上の実現 ③計画的ローテーションによる社員育成の実現	①売上/利益等の目標未達 ②利益未達にともなう研修未実施 ③お客様からのローテーション未承諾による配置の長期化	①お客様ニーズに最適な付加価値提供の遅れ/認識不足があったこと。対策のためのスタッフ工数を確保できず，現場内での細々とした対策で効果がなかったこと。 ②研修費を利益補填とすることを公認していたこと。 ③配置の長期化をやむなしとしたこと。	・社員の活性化をベースにした保守業務プロセス改善の実現(お客様を巻き込む事を含む) ・ツール導入による品質/生産性向上の実現 (会社で導入し，利用料を請請求する等の策を含む)
社員 (本人)	一部の社員が自分だけでは解決できない悩みを抱えている。 ①「出来たら開発部門等の評価されやすい部門に異動したい。」 ②「しかし，オンサイト勤務が長期化し，同じ職場/システム環境であったため， オンサイトの固有技術しかなく，また新たな職場での人間関係にも不安がある。」	①公正/明確な評価機会の確保 ②スキル向上のために組織による研修受講機会の確保，および計画的なローテーションの実現	①社内評価に偏りがあると感じている。 ②自分の仕事に対する自信喪失感がある。	組織として， ①新規開発等の見えやすい部門の評価に偏ったこと。 オンサイト/長期間の職場の評価の仕組み無し。 ②オンサイトへの長期間配置にともなう，技術スキルの偏り，コミュニケーションスキル不足への未対応 ※結果，社員が総じて「依存型思考」となっている。	まず社員の意識改革が先 ②マイビジョンをベースとした「ブレないための自分像を文書化する」 ・マイビジョン設定/文書化 ・「自己学習計画策定」 ・自習の実施 ↓ 仕事で成果を出す。 ↓ お客様/会社から認められる。 ↓ 計画的ローテーションの対象となる。 ①評価機会が増えた時に対象となることができる。

（文責：大島）

２．ＳＷＥＢＯＫ　Ｖ３（ＤＲＡＦＴ版）　第５章　ソフトウェア保守

IEEE Computer Society は，ソフトウェアエンジニアリング基礎知識体系（SWEBOK V3）のレビューコメントを公募ている（http://computer.centraldesktop.com/swebokv3review/）。前回の 2004Version から様々な点が改訂されいるが，ソフトウェア保守に関する章も次の改訂がなされた。

・「ソフトウェア保守における主な課題」知識領域の「管理上の課題」サブ領域の「外部調達」項目が「外部・海外調達」に変更

・「ソフトウェア保守の技法」知識領域に，「移行」「廃棄」のサブ知識領域が追加

・「ソフトウェア保守ツール」知識領域が追加

・IEEE/ISO/IEC 14764 2006，IEEE/ISO/IEC 24765 2010 等標準の改訂に伴う参考文献が統合（昨年取り組んだ A. April，A. Abran の書籍も紹介：さらなる文献　項番１）

当グループの有志がコメントの提出を目指して翻訳・理解を試みた。以下その結果を紹介する。

第５章　ソフトウェア保守

頭字語

KA Knowledge Area （**知識領域**）
MR Modification Request （**変更要求**）
PR Problem Report （**問題報告**）
SCM Software Configuration Management （**ソフトウェア構成管理**）
SLA Software Level Agreement （**サービスレベル合意書**）
SQA Software Quality Assurance （**ソフトウェア品質保証**）
V&V Verification and Validation （検証と**妥当性確認**）

序論

ソフトウェアの開発の作業は利用者の要求を満たすソフトウェア製品の引き渡しで終了する。それゆえ，ソフトウェア製品は変更，成長しなければならない。運用に入ると，欠陥が露呈したり，運用環境が変わったり，利用者の新しい要求が表面化する。ライフサイクル上保守段階は，瑕疵担保期間，すなわち稼働後の支援提供基幹が過ぎると始まるが，保守アクティビティはもっと前から発生する。

ソフトウェア保守はソフトウェアライフサイクルの必須部分の一つである。しかし，歴史的に見れば，これまで他の部分と同等の注目を得られなかった。これまで多くの組織体ではソフトウェア開発がソフトウェア保守よりもよりはるかに高い評価を得てきた。今その状況が変わり，組織体はできるだけ長くソフトウェアの運用を維持することによって，ソフトウェアの開発投資を最小化する努力をするようになった。オープンソースの考え方

によって，他が開発したソフトウェア成果物を保守する行為がさらに注目されるようになった。

このガイドでは，ソフトウェア保守を，ソフトウェアに対して費用対効果の高いサポートを提供するために必要なアクティビティの全体と定義する。これらのアクティビティは引き渡し後のステージと同様に引き渡し前のステージでも実施されている。引き渡し前のアクティビティは引き渡し後の運用，保守性，引き継ぎアクティビティの支援の決定の計画を含む（IEEE 2006,c6s9 参照)。引き渡し後のアクティビティはソフトウェアの修正，要員育成，ヘルプデスクの運用・連携が含まれる。

ソフトウェア保守の知識領域はソフトウェア工学における他のすべての側面と関係がある。そのため，その知識領域の記述にはこのガイドの他の章への参照をもつ。

ソフトウェア保守におけるトピックスの要素分け

ソフトウェア保守の要素分けを図１に示す。

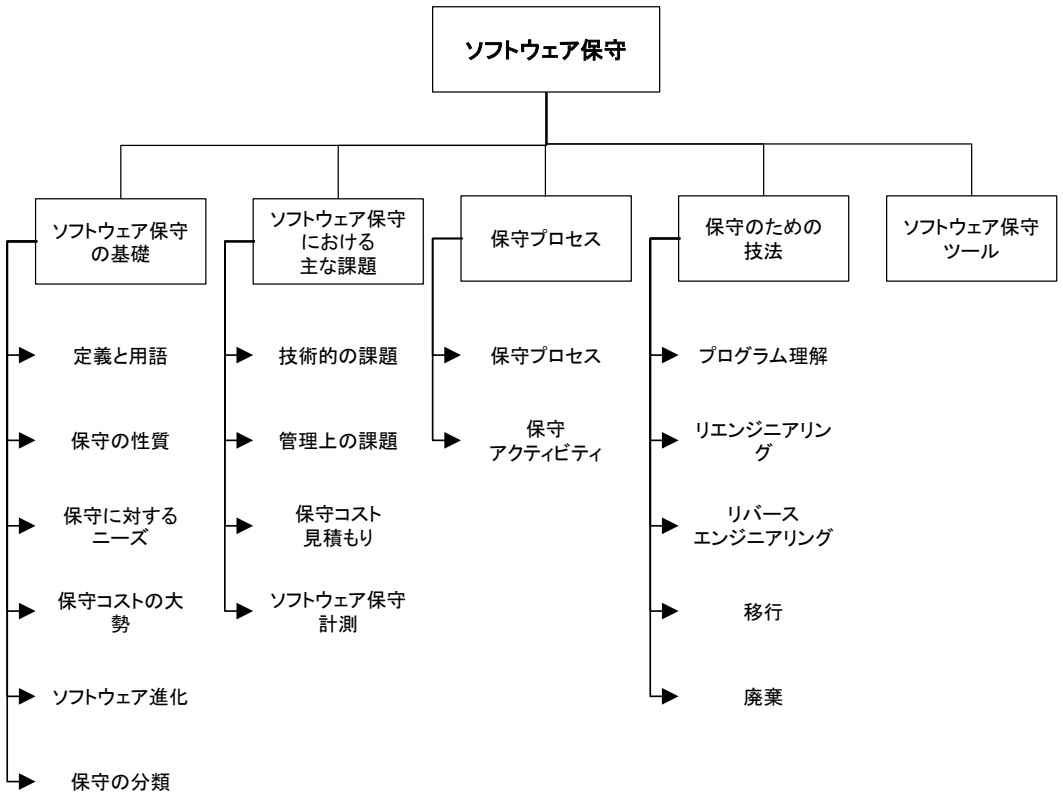


図1　ソフトウェア保守におけるトピックスの要素分け

１．　ソフトウェア保守の基本事項

本節はソフトウェア保守の役割と範囲を理解するために次に示す基本的な形でコンセプトと用語を紹介する。保守の定義を説明し，なぜ保守が必要なのかを強調する。ソフトウェア保守の種類は基本的な意味を理解する上で必須なものである。

１．１　定義と用号（1，c1s2，c2s2，c3s2；2，c3）

ソフトウェア保守の目的はソフトウェア保守の国際規格 ISO/IEC/IEEE14764 に定義されている（注１）。簡潔に言うと，ソフトウェア工学の背景から，ソフトウェア保守は多くの技術的プロセスの一つである。

ソフトウェア保守の目標は完整性を保持している既存ソフトウェアを修正することである。また，この国際規格はソフトウェアを最終引き渡しする前のいくつかの保守アクティビティを実施することの重要さを述べている。特に，同規格では引き渡し前の側面ではたとえば保守計画の重要さを強調している。

１．２　保守の性質(1，c1s3)

ソフトウェア保守はそのライフサイクル（開発から運用まで）を通してソフトウェア製品を支える。修正依頼は記録され，追跡され，その変更による影響が確認され，プログラムコードやその他のソフトウェア成果物が修正され，テスト実施が指揮され，ソフトウェア製品の新版がリリースされる。さらに，教育と日々の支援が利用者に提供される。

保守者は保守アクティビティを実施する組織体を意味する。この知識領域では,「開発者」という語とは対比して,「保守者」は保守アクティビティを実行する個人を指す場合がときどきある。

IEEE14764 では，ソフトウェア保守プロセスの主アクティビティとして，プロセス実装，問題分析及び修正分析，修正の実施，保守レビュー及び受入れ，移行，廃棄となっている。これらのアクティビティは 3.2 保守アクティビティで述べる。

保守者は開発者からソフトウェアの知識から学ぶことができる。開発者と接触し保守者が早めに関わると保守の苦労を少なくする助けとなる。ときには，初期開発者に連絡できなかったり，他の業務に異動していたりすることがあり，このような場合には，保守者とって追加的な課題が発生することになる。保守は，開発成果物（例：コード，ドキュメント）を受け取り，直ちにそれらの支援を提供し，ソフトウェアのライフサイクルを通して，徐々に進化／保守する必要がある。

１．３　保守に対するニーズ

保守は，ソフトウェアが利用者要求を継続して満たすように保証するために，必要とされる。保守はいくつかのソフトウェアライフサイクルモデル（例：スパイラル）を使って開発されたソフトウェアにも適用される。ソフトウェアは，ソフトウェアに対する是正，または非是正アクションよって変更される。保守は次のことを実施する必要がある

・障害の是正
・設計の改善
・エンハンスの実施
・他のソフトウェアとのインタフェースをとる

・別のハード，ソフト，システム要素，通信機能を使えるようにプログラムを適合させる
・レガシーソフトウェアの移行
・ソフトウェアの廃棄

保守者のアクティビティは次の５の重要特性からなる。
・ソフトウェアの日々の動きの管理を維持する
・ソフトウェア修正の管理を維持する
・既存機能を完全なものにする
・セキュリティ上の脅威を認識し，セキュリティ上の脆弱性を除去する
・ソフトウェアの性能が，容認できないレベルまで劣化することを防止する

１．４　保守コストの大勢(1, c4s3, c5s5.2)

保守は，ソフトウェアのライフサイクルにおける財政的な資源の大部分を消費する。ソフトウェア保守に対する一般的な見方では，それは単なる障害の除去である。しかし長年の研究や調査は，ソフトウェア保守の大部分（80%超）が非是正アクションに使われることことを示している(1, figure 4.1)。改良と訂正を一緒くたにした管理報告では，訂正に多くのコストを使っているという勘違いに陥りやすくなります。ソフトウェア保守の種類を理解することは，ソフトウェア保守コスト構造を理解する上で助けになる。また，ソフトウェアの保守性に影響する要素を理解することは，コストを抑える助けにできます。グラブとタカンは次のようにある種の環境要因とソフトウェア保守コストとの関係を示している。
・運用環境はハードとソフトに関係する
・組織環境は理念，競合相手，プロセス，製品，要員に関係する

１．５　ソフトウェアの進化(1, c3s5)

レーマンは 1960 年代末にはすでにソフトウェア保守と進化を論じた。20 年以上に渡るレーマンの研究は８つの「進化の法則」の定義を導きました。レーマン主要な発見は「保守は進化のための開発である。また，長い間にソフトウェアに何が起こるのかを理解することは，保守の意思決定の役に立つ」と提案した。別の意見は「特別な入力（または規制）が存在することを除けば，保守は継続的な開発である」，別の言い方として「既存の大規模ソフトウェアは決して完成することがなく，進化し続ける。進化するにつれて，何らかのアクションをとらなければ複雑さは増加していくので，これらを低減させなければならない」。

１．６　保守の分類(1, c3s3.1; 2,c3, c6s2)

リンツとスワンソンは保守を三つの分類（タイプ）として定義した（是正保守，適応保守，完全化保守）(1, 223 c4s3)。その後，IEEE14764 で次の四つの分類に変更された。

・是正保守：ソフトウェア製品の引き渡し後に発見された是正すべき問題に対する事後の修正（修理）のこと。この分類には緊急保守（是正保守がなされるまでソフトウェアの運用を維持するために，一時的になされる非計画的修正）を含む。

・適応保守：環境の変化が発生または進行中にソフトウェア製品が使い続けられるように行うソフトウェア製品化の引き渡し後の修正のこと。(例) OS がアップグレードされ，そのソフトウェアに変更が必要になった場合

・完全化保守：利用者の改良要件，文書の改善，ソフトウェアの性能，保守性，他のソフトウェア属性の測定と改善を提供すねソフトウェア製品引き渡し後の修正のこと。

・予防保守：運用上の障害となる前にソフトウェア製品で潜在する障害を発見し，是正するソフトウェア引き渡し後の修正のこと。

IEEE14764 は適応保守と完全化保守を改良保守とし，是正保守と予防保守を訂正という分類に入れている（次表１参照)。

表１　ソフトウェア保守の分類

	訂正	改良
プロアクティブ（事前実施）	予防保守	完全化保守
リアクティブ（事後応答）	是正保守	適用保守

２．ソフトウェア保守の主要課題

ソフトウェアの効果的なメンテナンスを保証するために，多くの重要な課題を取り扱わなくてはならない。 ソフトウェア保守がソフトウェア・エンジニアにユニークで技術的な，そして管理上の挑戦課題を提供することを理解することは重要であう。例えば，別のソフトウェア・エンジニアが開発した 500K コード行にもおよぶソフトウェアに欠陥を見いだそうとするのは良い例である。 同様な例に，常にソフトウェア開発者と資源を競い争わなければならないということがある。 しばしば次のリリースのコーディングをし，現在のリリースのために緊急パッチを送っている間に，未来のリリースを計画することは，同じく挑戦的課題のである。 次のセクションはソフトウェア保守と関係がある技術的な，そしていくつかの管理上の課題を述べる。 それらは次のトピック見出しの下に分類した：

・記述的課題
・管理上の課題
・コスト見積もり
・計測

２．１　技術的課題

２．１．１　限られた理解(1, c6)

限られた理解は，ソフトウェア・エンジニアが，彼あるいは彼女が開発しなかったソフトウェアでどこを変更あるいは，訂正にするべきかについて，どれだけ早く理解できるかに関係している。研究によれば，全てのメンテナンスの努力のおよそ半分が修正されるソフトウェアを理解することに費やされることを示している。 それで，ソフトウェア理解の項目はソフトウェア・エンジニアにとって大きな関心事になっている。 理解はテキスト指向の表現のためにいっそう困難となっている － 例えば，ソースコードであれば，よくあるようにもし変更が文書化されず，もし開発者がそれを説明するために利用可能ではないなら，そのリリース / バージョンを通してソフトウェアの進化過程を追跡することは大変難しい。したがって，ソフトウェア・エンジニアが初めにソフトウェアの限られた理解しか持っていないかもしれず，これを改善するためには大変な努力を払わなければならない。

２．１．２　テスト(1, c9; 2, c5s2.2.2)

ソフトウェアの主要部の上に全てのなテストを繰り返すコストは時間と金銭的に重大である。要求された問題報告が正当であることを証明するために， 保守者は，再現し，あるいは適切なテストを行なうことによって問題を実証するべきである。回帰テスト（修正が思いがけない影響を引き起こさなかったかを確認するためにソフトウェアあるいはコンポーネントを選択的に再びテストすること）は保守にとって重要なテストの概念である。さらに，テストをするための時間を見つけることはしばしば困難である。異なった保守チームのメンバーが同時に異なった問題に取り組んでいるときには，テストを実施しなければならないという問題がある。ソフトウェアが重要な機能を実行している場合，テストのためにそれをオフラインにすることは難しい。ソフトウェアテスト KA では，回帰テストのサブトピックで，この問題に関して追加の情報とリファレンスを提供している。

２．２．３　影響分析(1, c13s3; 2, c5s2.5)

影響分析は，既存ソフトウェアの変更の影響について完全な分析を，どのように費用効果的に行なうかを記述する。保守者はソフトウェアの構造と内容の詳細な知識を持っていなければならない。彼らはその知識を，ソフトウェア変更のリクエストに影響されるすべてのシステムとソフトウェアプロダクトを識別して，そして変更を達成するために必要なリソースの見積もりを検討する，インパクト分析を実行するために使う。さらに，変更をするリスクが決定される。 変更の要求は，時々変更要求（MR）と呼ばれて，そしてしばしば問題報告（ＰＲ）とも呼ばれて，最初に解析されて，そしてソフトウェアの項目に変換されなくてはならない。 変更の要求がソフトウェア構成管理プロセスに入力した後で，影響分析が行なわれる。 IEEE 14764 影響分析のタスクを述べる：

・変更要求・問題報告の分析
・問題の再現と検証
・修正を実施するための開発オプション
・変更要求・問題報告，結果，実行オプションの文書化
・選択された修正オプションのための認可の獲得

問題の重度はしばしば問題がどのように，そしていつ修正されるかを決めるために使われる。その後，ソフトウェア・エンジニアは影響を受けるコンポーネントを識別する。いくつかの可能性がある解決が提供されます，そして次に最も良い行動方針が推薦される。

保守性を念頭に置いて設計されたソフトウェアが影響分析を極めて容易にする。もっと多くの情報がソフトウェア構成管理 KA で見つけることがでる。

２．１．４　保守性(1, c12s5.5; 2, c6s8)

開発の間に保持性の問題をどのように奨励して，どこまでも追及すればよいか？ IEEE 14764（２， c3s4）は保守性を修正されるソフトウェアプロダクトの能力と定義します。 修正は訂正，機能の改良あるいは環境や要求事項の変化に対するソフトウェアの適応を含むであろう。 保守性はソフトウェアの主な品質特性の１つである。保守コストを減らすために保守性特性が指定されて，再検討されて，ソフトウェア開発活動の間に制御されなければならない。もしこれがうまくされるなら，ソフトウェアの保守性は良くなるだろう。しかし，これは達成することが困難な場合も多い。なぜならソフトウェア開発のプロセスにおいて，保守性副特性は注目がされていないからである。開発者は多くの他の活動により夢中になっていて，そしてしばしば保守者の必要事項を無視することになる。これはソフトウェアドキュメンテーションとテスト環境の欠如をもたらすかもしれない，そしてしばしばそうなる，その結果その後プログラム理解と影響分析における困難の主な原因となる。組織的な，そして成熟したプロセス，技術とツールの存在がソフトウェアの保守性を強化する助けとなる事実も観察されている。

２．２　管理上の課題

２．２．１　組織目標との整合性(1, c4)

組織的な目標は，ソフトウェア保守活動が投資と回収をどのように説明するかを記述する。最初のソフトウェア開発は通常プロジェクトベースで，定義されたタイムスケールと予算がある。主な主眼点は利用者の必要を満たす，時間通りに予算の範囲内で中，プロダクトを引き渡すことである。それと対照的に，ソフトウェア保守はしばしば，できる限り長い間ソフトウェアの寿命を延ばす目的を持っている。さらに，それはソフトウェア更新と改良に対する利用者要求に適合する必要によって動かされることもある。両方のケース

では投資回収は明確ではない。上級管理層の見方はしばしば，組織にとって量的な利益のない，重要な資源を消費する主なアクティビティである。

２．２．２　人員確保(1, c4s5, c10s4)

人員確保はソフトウェア保守人員を引き付けて，そしてその状態を保つ方法を示す。保守はしばしば魅惑的な仕事だと見られていない。結果として，ソフトウェア保守人員がしばしば「２級市民」だと見なされ，そしてその結果士気が損なわれている。

２．２．３　プロセス(1, c5; 2, c5)

ソフトウェアライフサイクルプロセスはソフトウェアおよびそれに関連するプロダクトを開発して，そして保守するための人々が使うアクティビティ，方法，プラクティス，変形をまとめた集合体である。プロセスレベルでは，ソフトウェア保守アクティビティはソフトウェア開発アクティビティと共有するものが多い（例えば，ソフトウェア構成管理は両者にとって欠くことの出来ないアクティビティである)。 保守はさらにソフトウェア開発に見いだされないいくつかのアクティビティを要求する（詳細なユニークなアクティビティは，セクション３．２参照)。 これらのアクティビティが管理への課題となる。

２．２．４　保守の組織的側面(1, c10; 2, c7s2.3)

組織的側面は，ソフトウェア保守の責務を担当させる組織そして／あるいは職務をどうのようにして決めるかを記述する。 ソフトウェアを開発したチームは，それが一度運用にはいると，ソフトウェアを保守することを必ずしも割り当てられる訳ではない。

ソフトウェア保守の職務をどこに置くかを決めることに関しては，ソフトウェアエンジニアリングの組織が，例えば，最初の開発者のところに留まるか，あるいは恒常的な保守に割り当てられたチーム（あるいは 保守担当者）に行くかもしれない。恒常的な保守チームを持っていることは多くの利点がある：

- 専門化を可能にする
- コミュニケーション経路を作る
- エゴレスで，学究的な雰囲気の促進
- 個人に対する依存（属人化）を減らす
- 定期監査チェックを可能にする

それぞれのオプションに多くの有利な点と不利な点があるので，決断は事例ごとにされるべきである。重要なことは，組織の構造にかかわらず，単一つのグループあるいは個人への保守責任の割り当てと委任である。

２．２．５　外部・海外調達

外部調達と海外でのソフトウェア保守は主要産業になった。組織は，ソフトウェア保守を含めて，ソフトウェアの職全体を外部調達しつつある。多くの場合，組織が基幹ビジネスで使われるソフトウェアのコントロールを失うことを好まないため，外部調達のオプションはそれほど致命的でないソフトウェアに選択される。外部受託者の主な課題の一つは，要求される保守サービスの範囲，サービスレベル合意の項目，契約書の詳細を決定することである。外部受託者は保守のインフラに投資する必要があり，遠隔サイトのヘルプデスクは母国語の話し手が配置されるべきである。外部調達は有意義な初期投資，および自動化を要求する保守プロセスの確立を必要とする。

２．３　保守コストの見積もり

ソフトウェア保守のためのコスト見積もりの問題を扱うために，ソフトウェア技術者は，先に述べたように，ソフトウェア保守には様々な分類がある，と言うことを理解しなければならない。目的を計画することに対して，コストを見積もることはソフトウェア保守の重要な側面である。

２．３．１　コスト見積もり(1, c7s2.4)

セクション２．１．３で述べたように，影響分析がソフトウェア変更要求によって影響を受けている全てのシステムとソフトウェアプロダクトを識別し，変更を達成するために必要なリソースの見積もりを作成する。

保守のコスト見積もりは多くの技術的な，そして技術的でない要因によって影響を受けている。 IEEE 14764 は「ソフトウェア保守に対するリソースを見積もることへの２つの最も一般的なアプローチが，統計モデルの使用と経験値の使用である」（２， c7s4.1）と述べている。これら２つを組み合わせもよく使われる。

２．３．２　パラメトリックモデル(1, c12s5.6)

統計コストモデリング（数学的なモデル）をソフトウェア保守に適用することにおいて，いくつかの研究が着手されました。重要なことは，過去の保守からの過去データが，数学的なモデルを使い，計測するために必要とされる，ということである。コスト中心の属性が見積もりに影響を与える。

２．３．３　経験値(1, c12s5.6)

経験が，専門家の判断のかたちで，しばしば保守の労力を見積もるために使われる。明らかに，保守見積もりへの最も良いアプローチは過去のデータと経験を組み合わせることである。修正を行うコスト（人々の数と時間の量に関して）がそれから得られる。保守見積もり過去データが測定プログラムの結果として提供されるはずである。

２．４　ソフトウェア保守計測

フェントンは計測可能の属性をもつソフトウェア保守と関係がある３つのエンティティーを見いだした：プロセス，リソースとプロダクト（１， c12s3.1)。 ロバート・グレイディは保守のための会社規模のソフトウェア測定プログラムを提出した。それには，ソフトウェア保守測定フォームとデータ収集が記述されている（１)。

ソフトウェア，保守プロセス，人員の属性から得ることが出来るいくつかのソフトウェアの計量尺度がある。グラブとタカンが規模，複雑度，品質，理解性，保守性，労力をリストアップした。これらの計量尺度は，保守だの測定プログラムのための良い出発点を構成する。プロセスおよびプロダクトの測定の議論はソフトウェアエンジニアリングプロセス知識領域で提示される。ソフトウェア測定プログラムの項目は，フトウェアエンジニアリング管理知識領域で記述される。

２．４．１　特有な計量尺度(1, c12)

保守者はどの計量尺度がその組織の自身の状況に基づいて彼の特定の組織にふさわしいか決定しなくてはならない。ソフトウェア品質モデルはソフトウェア保守に特定の計量尺度を示唆します。そのリストは保守性の４つの副特性のそれぞれのために多くの基準を含みます。

・分析容易性：欠陥あるいは不具合の原因を診断しようとするか，あるいは修正される部分を識別するために費やされる保守者の労力あるいは資源の計量尺度。
・変更容易性：指定された修正を実行することに関係する保守者の労力の計量尺度。
・安定性：テストの間に遭遇するものを含めて，ソフトウェアの意外な行動の計量尺度。
・テスト容易性：修正されたソフトウェアをテストするために必要な保守者と利用者の努力の計量尺度。

グラブとタカンも保守者使用するその他の計量尺度を示している。

・ソフトウェアの規模
・ソフトウェアの複雑度(McCabe and Halstead)
・理解性
・保守性

異なったアプリケーションに，分類に応じて，ソフトウェア保守の努力を提供することは利用者と彼らの組織に経済情報を提供します。それはソフトウェア保守プロフィールを測って，そして内部に組織と比較することができる。

３．　保守プロセス

保守プロセスは，IEEE 14764 で記述された。標準的なソフトウェア・エンジニアリン

グ・プロセスとアクティビティのほかに，保守者に特有な多くのアクティビティがある

３．１　保守プロセス(1, c5; 2, s5)

保守プロセスは必要とされるアクティビティとそれらのアクティビティの詳細なインプット / アウトプットを提供する；それらはソフトウェア保守国際規格 IEEE 14764 記述されている。 IEEE 14764 の保守プロセスアクティビティを図２に示す。IEEE 14764 のソフトウェア保守アクティビティは次を含む

・プロセス実装
・問題分析及び修正の分析
・修正の実施
・保守レビュー及び受入
・移行
・廃棄

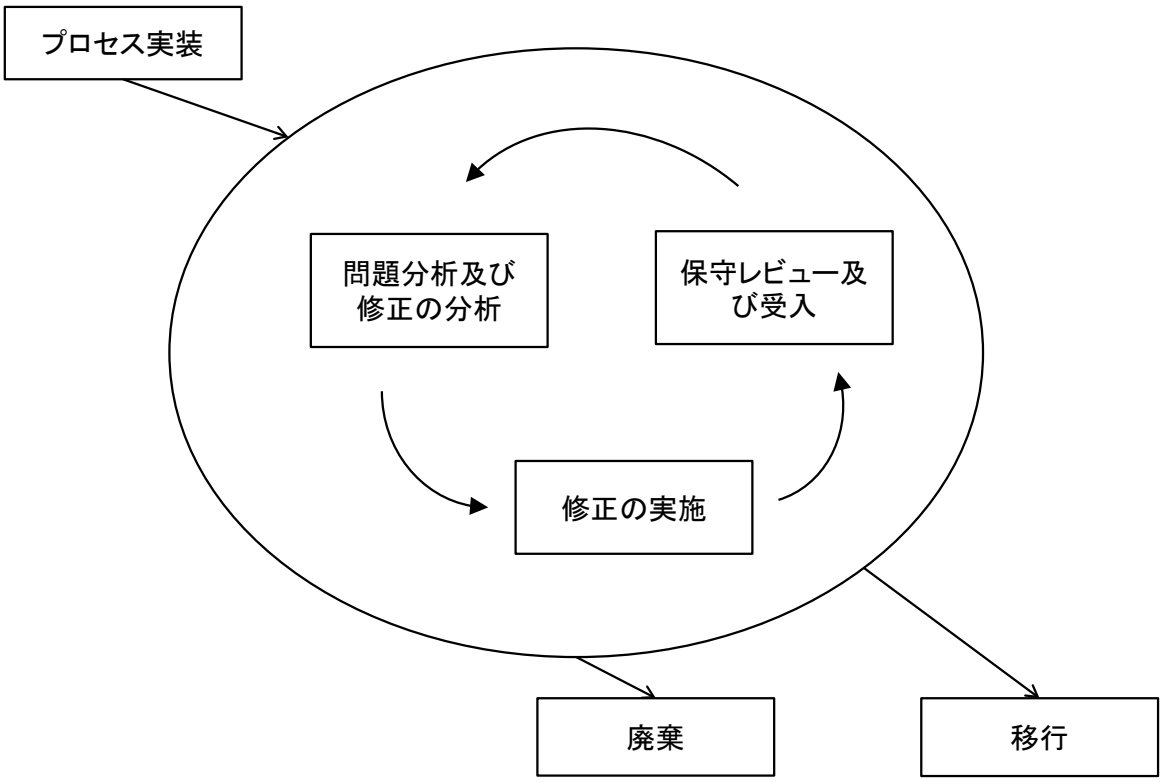


図2.　ソフトウェア保守プロセス

グラブとタカンは，多数の保守プロセス・モデルを記述した。

・迅速修正（quick　fix？）
・Boehm’s
・Osborne’s
・反復改良
・リユース指向

最近，軽いプロセスを促進するアジャイル方法論が保守のために改造された。この要求項目は保守サービスの速いターンアラウンドに対する常に増大する要求から出現する。ソフトウェア保守プロセスへの改良が専門的なソフトウェア保守能力成熟度モデルによってサポートされた，ソフトウェア保守成熟モデル（April と Abran 2008）と是正保守成熟モデル（Kajko-Mattsson　2001）はソフトウェア 保守者の独特なプロセスとアクティビティを扱う。

３．２　保守アクティビティ(2，c5，c6s8.2，c7s3.3)

保守プロセスは，既存のソフトウェアプロダクトがその完整性を維持している間に，それを修正するために必要なアクティビティとタスクを含んでいる。これらのアクティビティとタスクと仕事は保守者の責務である。すでに述べたとおり，多くの保守アクティビティはソフトウェア開発のものに類似している。保守者は分析，設計，コーディング，テストおよび文書化を行なう。

彼らはアクティビティの要求項目そ追求しなければならない（開発におけるとてお同じ様に)，そして，ベースラインが変化するとき，文書を更新しなければならない。IEEE 14764 は保守者が開発プロセスを使うとき，彼がそれを自分に特定されたニーズに合うように適応させることを勧めている（２，　c5s3.3.2)。しかしながら，ソフトウェア保守では，いいくつかのアクティビティはソフトウェア保守なプロセスを含んでいる。

３．２．１　特有のアクティビティ(1，c6,c7；2，c3s10，c6s9，c7s2-c7s3)

ソフトウェア保守に特有な多くのプロセス，アクティビティとプラクティスがある。

・プログラム理解：ソフトウェアが何をするか，そしてどの部分が一緒に動くかの全体的な知識を得るために必要とされるアクティビティ。
・移行：ソフトウェアが開発者から保守者に前進的に引き渡される制御された，整理されたアクティビティのシーケンス。
・変更要求の受付／リジェクト：変更要求の仕事量があるサイズ，労力，複雑さを超えると，保守者によりリジェクトされ開発者に差し戻される

・保守ヘルプデスク：エンド利用者と保守者を調整する支援組織。それは，評価，優先順位付け，変更要求のコスト見積もりを調整する
・影響分析：起こりうる変更により影響を受ける領域を識別する技術
・保守サービスレベル合意，保守ライセンス，契約：サービスと品質目標を記述する契約合意

３．２．２　支援アクティビティ(1，c9；2，c4s1，c5，c6s7)

保守者も文書化，ソフトウェア構成管理，検証と妥当性検査，問題解決，ソフトウェア品質保証，レビューと監査のような，支援アクティビティを行だろう。もう１つの重要な支援アクティビティは保守者と利用者を訓練である。

３．２．３　保守計画アクティビティ(2，c7s3)

ソフトウェア保守の重要なアクティビティの一つが計画を立てることである。保守者は，計画の見地と結び付けられる多くの課題に取り組まなければならない。

・事業計画（組織レベル）
・保守計画（移行レベル）
・リリース／バージョン計画（ソフトウェアレベル）
・個別のソフトウェア変更要求計画（要求レベル）

個別の要求レベルでは，計画が影響分析の間に行われる（詳細は，２．１．３章　影響分析参照)。リリース／バージョン計画アクティビティは保守者に次を要求する

・個々の要求が有効になる日付の収集
・継続するリリース／バージョンの内容を利用者と合意
・起こりうる対立を識別し，代案を作成すること
・与えられたリリースのリスクを評価し，問題が起こった場合に備えてバックアップ計画を作ること
・全てのステークホルダーに広報すること

一般に，ソフトウェア開発プロジェクトが数カ月から数年まで終えたとしても，保守フェーズは通常，何年もの間続く。リソースの見積もりは保守計画の主要な要素である。ソフトウェア保守計画は新しいソフトウェアを開発するという決定から始り，そして品質目標を考慮しなければならない。コンセプト文書がメンテナンス計画に続いて，作られるべきである。個々のソフトウェアの保守コンセプトは計画にかかれる必要があり，次の項目を扱うべきである

・ソフトウェア保守の範囲
・ソフトウェア保守プロセスの適用
・ソフトウェア保守組織の定義
・ソフトウェア保守コストの見積もり

次のステップは相当するソフトウェア保守計画を作ることである。この計画はソフトウェア開発の間に準備されなければならず，そして利用者がどのようにしてソフトウェア修正あるいは問題報告要求するかを明示しなければならない。ソフトウェア保守計画が IEEE 14764 であつかわれている。それは保守計画にガイドラインを提供している。

最終的に，最も高いレベルで，保守組織は組織のすべての他の部門とまったく同じように事業計画アクティビティ（予算上の，金融の，そして人間の資質）を行なわなければならない。そうするために必要とされるマネージメント知識は章「ソフトウェアエンジニアリングの関連した訓練」を参照。

３．２．４　ソフトウェア構成管理(1，c11；2，c5s1.2.3)

IEEE 14764 はソフトウェア構成管理を保守プロセスの重要な要素の一つだと記しているソフトウェア構成管理の手続きは，ソフトウェアプロダクトの識別，認可，実装，リリースに要求されるそれぞれのステップの検証，妥当性検査，識別を規定しなければならない。

修正の要求あるいは問題報告を単純に追跡するだけでは十分ではない。ソフトウェアプロダクトとそれに加えられたどんな変更でも制御されなくてはならない。制御は，認可されたソフトウェア構成管理 (SCM) プロセスを実装し，強制することによって，確立される。ソフトウェア構成管理知識領域は SCM の詳細を規定されており，ソフトウェア変更要求が登録されて，評価されて，そして承認されるプロセスを論じる。ソフトウェア保守のための SCM は変更ソフトウェア開発のための SCM と小さな点で数多く違っていおり，それは稼動しているソフトウェアの上で制御されなければならない。SCM プロセスは，構成管理計画と運用手順を作成し，そしてそれに従うことによって，実装される。保守者が次のリリース / バージョンの内容を決定するために構成管理委員会に参加する。

３．２．５　ソフトウェア品質管理(1，c12s5.3；2，c6s5，c6s7-c6.8)

ソフトウェアの保守の結果，品質が良くなることを単純に希望するだけでは十分ではない。保守者はソフトウェア品質プログラムを持つべきである。保守プロセスを支援することあは，計画され，実装されたプロセスでなければならない。望ましい品質を達成するために，ソフトウェアの品質保証（SQA）のためのアクティビティと技法，つまり，Ｖ＆Ｖ，レビューと監査はすべての他の全てのプロセスと連携して選なくてはならない。保守者がソフトウェア開発のプロセス，技法，配布可能物（例えば，テストドキュメント)，テスト結果を改造することは推奨される。もっと多くの詳細がソフトウェア品質知識領域 に見いだされる。

４．　保守のために技法

この副領域では，ソフトウェア保守において利用される一般に認められた幾つかの技術を紹介する。

４．１　プログラム理解(1，c6，c14s5)

プログラマは，変更を実施するため，プログラムを読込んだり，理解したりするのに，相当な時間を費やす。Code Browser（※高橋注　特定のツールのことか？http://tibleiz.net/code-browser/）は，ソースコードを分類（整理）したり，表示したりするために利用される，プログラムを理解するための重要なツールである。明確で，簡潔な文書もまた，プログラムの理解を助けるだろう。

４．２　リエンジニアリング(1，c7)

リエンジニアリングは，ソフトウェアを新しい形式に再構成するために行うソフトウェアの調査と改造であると定義さ，続けて実施する新しい形式の実装を含む。リエンジニアリングは，保守性を改善するために試みられるのではなく，老化したレガシーなソフトウェアを置換するために行われる。

リファクタリングは，その振る舞いを変更せずに，プログラムを再編成することに目的としたリエンジニアリング技術である。それは，プログラムの構造とその保守性の改善を目指す。リファクタリングの技法は，マイナーチェンジの中で，利用することができます。

４．３　リバースエンジニアリング(1，c7，c14s5；2，c6s2)

リバースエンジニアリングは，ソフトウェアのコンポーネントおよびそれらの相互関係を識別し，別の形式やより高い抽象概念でのレベルのソフトウェア表現ために行う，ソフトウェアを分析するプロセスである。リバースエンジニアリングは，受動的である。何故なら，ソフトウェアを変更せず，新しいソフトウェア生成するものではない。リバースエンジニアリングにより，ソースコードからコールグラフおよび制御フローラフを作成される。リバースエンジニアリングの１つのタイプは，再文書化である。別のタイプは，設計回復である。最後に，データ・リバースエンジニアリングは，物理的データベースから論理スキーマを回復しようとするもので，ここ数年その重要性が増してきた。ツールは，リバースエンジニアリングや，再文書化や設計回復のような関連するタスクにとっての重要なものである。

４．４　移行(2，c5s5)

ソフトウェアは，その生涯の中で，異なる環境において稼働させるための修正をしなければならないだろう。ソフトウェアを新しい環境に移行させるために，保守者は，移行を遂行するために必要なアクションの決定を必要とし，移行の要求項目，移行ツール，プロ

ダクトおよびデータの変換，事項，評価，サポートをカバーする移行計画に移行をもたらすために要求されるステップを作成し，文書化する必要がある。ソフトウェアの移行は，多くのアクティビティ（活動）を必要とする。

・目的の通知：何故もう古い環境をサポートすることが出来ないのかの主張，それに続いて，更に新しい環境の記述とそれが有効となる日付の記述
・平行運用：古い環境と，新しい環境の利用可能にし，その結果，利用者は，新しい環境へのスムーズな転換（変遷）を体験する
・完成の通知：予定された移行が完了した時，通知を全ての関係者に送られる
・ポストオペレーション・レビュー：平行運用と，新しい環境に変わる影響の評価
・データの保管：古いソフトウェアデータの格納

４．５　廃棄(2，c5s6)

一旦，ソフトウェアがその耐用年数の終了に到着したならば，それは廃棄されなければならない。廃棄決定を下すのを支援するために，分析が実施されるだろう。この分析は，廃棄計画に含まれており，廃棄計画は廃棄要求事項や，影響，置換，スケジュールおよび成果をカバーしています。

同様にデータのアーカイブコピーのアクセシビリティも含まれるだろう。ソフトウェアの廃棄は，移行に似て，多くのアクティビティを必要とする。

５．ソフトウェア保守ツール(1，c14)（2，c6s4)

このトピックスは，既存のソフトウェアを修正するソフトウェア保守において，特に重要なツールを網羅している。プログラム理解とリバースエンジニアリングは，主な保守タスクであり，ツールは有用である。ツールは人間のプログラムの理解を支援することができる。例を，次に示す。

・プログラムスライサー：プログラムの変更によって影響を受けた部分だけを選択する。
・静的アナライザ：プログラムの概要や，内容の要約を可能にする
・動的アナライザ：保守者が，プログラムの実行経路をトレースすることを可能にする
・クロスリファレンス：プログラムコンポーネントの索引を生成する
・従属分析（依存性分析）：保守者が，プログラムコンポーネント間の相互関係を分析し，理解することを支援する

リバースエンジニアリングツールは，既存のプロダクトから仕様書や設計書のような成果物を創造する逆方向のプロセスを支援する。そして，それは，古いプロダクトから新しいプロダクトを生成しするように，変形されたのかも知れない。保守者は，さらにソフトウェアのテスト，構成管理，文書化，および測定ツールを使用する。

ソフトウェア保守に関する推奨文献

1. P. Grubb and A.A. Takang, Software Maintenance: Concepts and Practice, World Scientific Publishing: River Edge, NJ, 2003.

2. IEEE/ISO/IEC Std. 14764-2006, "Software Engineering - Software Lifecycle Processes - Maintenance," IEEE, 2006.

3. IEEE/ISO/IEC Std. 24765, "Systems and Software Engineering - Vocabulary," IEEE, 2010.

4. H.M. Sneed, "Offering Software Maintenance as an Offshore Service," IEEE Int'l Conf. Software Maintenance, IEEE CS Press, 2008, pp. 1-5.

トピックスと参照文献の対照表

	(IEEE/ISO/IEC 24765 2010)	(IEEE/ISO/IEC 14764 2006)	(Grubb and Takang 2003)	(Sneed 2008)
1.ソフトウェ保守の基本事項				
1.1 定義と用語		C3	c1s2, c2s2	
1.2 ソフトウェア保守の性質			c1s3	
1.3 ソフトウェア保守の必要性			c1s5	
1.4 ソフトウェア保守のコストの大勢			c4s3,c5s5.2	
1.5 ソフトウェア進化			c3s5	
1.6 ソフトウェア保守の分類		c3, c6s2	c3s3.1, c4s3	
2.ソフトウェア保守の重要要素				
2.1 技術的要素				
制限された理解			c6	
テスト		c6s2.2.2	c9	
影響分析		c5s2.5	c13s3	
保守性		c6s8, c3s4	c12s5.5	
2.2 管理上の要素				
組織目標との調整			c4	

保守要員			c4s5, c10s4	
プロセス		c5	c5	
保守の組織的な側面		c7s.2.3	c10	
外部調達／海外				all
2.3 保守コスト見積もり				
コスト見積もり		c7s4.1	c7s2.4	
統計的モデル			c12s5.6	
経験			c12s5.5	
2.4 ソフトウェア保守計測		c6s5	c12, c12s3.1	
特有な計量尺度			c12	
3.保守プロセス				
3.1 保守プロセス	s5.5	c5	c5	
3.2 保守アクティビティ		c5, c5s3.3.2 c6s8.2, c7s3.3		
固有のアクティビティ		c3s10, c6s9, c7s2, c7s3	c6,c7	
支援アクティビティ		c4s1, c5, c6s7	c9	
保守計画アクティビティ		c7s2, c7s.3		
ソフトウェア構成管理		c5s1.2.3	c11	
ソフトウェア品質		c6s5,c6s7, c6s8	c12s5.3	
4.保守技法				
4.1 プログラム理解			c6,c14s5	
4.2 リエンジニアリング			c7	
4.3 リバースエンジニアリング		c6s2	c7, c14s5	
4.4 移行		c5s5		

4.5 廃棄		c5s6		
5. ソフトウェア保守ツール		c6s4	c14	

付録A．さらなる文献

1. A. April, A. Abran, Software Maintenance Management: Evaluation and Continuous Improvement, Wiley-IEEE Computer Society Press, 2008.

2. M. Kajko-Mattsson, "Towards a business maintenance model," IEEE Int'l Conf. Software Maintenance, 2001, pp. 500-509.

(訳, 増井, 高橋 (芳), 弘中　監修　高橋 (芳))

３．啓蒙・広報活動

本年次の啓蒙・広報活動は，SERC フォーラムを主催した。

3.1　SERC フォーラム：「大規模障害に学ぶソフトウェア保守」開催

3.1.1 コンセプト

【テーマ】

ソフトウェア保守者が見た　みずほ銀行オンライン障害

【説明】

私たちの生活を支える社会基盤としてソフトウェアシステムは無くてはならないものになり，ソフトウェアシステムの障害が与える影響は益々大きくなっています。ソフトウェアの障害を減らし，社会に与える影響を軽減するためには，過去の失敗から学び，教訓を得ることが重要あることは論を待たないでしょう。しかしながら，ソフトウェアシステムの障害の情報は当事者の企業に閉ざされ，一般に公表される事は希で，ソフトウェア業界が失敗から学び，発展する機会が失われることが多々ありました。

昨年の大震災の直後に発生したみずほ銀行のオンライン障害は，社会生活に大きな影響を与えた例であるとともに，システムの障害の経緯・原因などの詳細な情報が公表された数少ない例であり，ソフトウェア業界にとって貴重な情報得ることができした。当研究会でもこの情報をソフトウェア保守の発展に有効に活用しようと，分析，検討を進めてまいりました。本フォーラムでは，この研究で得られた知見や教訓を発表いたします。

3.1.2　プログラム

【日時】

2012 年 5 月 18 日（金）13:30-16:50（受付 13:00-）

【プログラム】

13:00 - 13:30　受付

13:30 - 13:35　開会の挨拶

13:35 - 13:55　みずほ銀行障害の概要

SERC 研究員　高橋　宏志

13:55 - 14:15　システム障害調査委員会の報告書　概説

SERC 研究員　増井　和也

14:15 - 15:00　SERC の提言１　組織論，マネジメントの観点から

SERC 研究員　弘中　茂樹

SERC 研究員　大島　道夫

15:00 - 15:30　SERC の提言２　ソフトウェア保守技術の観点から

SERC 研究員　高橋　芳広

15:30 - 15:40　休憩

15:40 - 16:40　質疑応答

16:40 - 16:50 クロージング

3.1.3　資料

※資料　１－２参照 (Page14)

（文責：高橋芳）

３．ま　と　め

第二十一年度活動を終えて
～幹事よりひとこと～

NTT データ CCS　馬場　辰男

企業活動に留まらず、産業や国家活動のほとんどが情報システムの支援によって遂行される時代、その情報システムの品質や保守の稚巧が、企業経営の命運を左右するばかりでなく、人命や国家の信頼にも決定的な影響を与える時代となり、ソフトウェアの保守は、運用を含めて科学的工学的に管理・対処されなければならなくなりました。

当研究会（SERC）は、このソフトウェアに付随する保守の問題を、開発標準、品質、用語、要員問題、プロセス、ツール等の種々様々な観点から研究してまいりました。また、その活動を通じて、ソフトウェア保守の JIS 規格化にも少なからず貢献できたと自負しております。

私達の研究活動の積み重ねは、既に 20 年を越え、成果の蓄積は相当なものになっておりますが、一方、実社会での情報システムの運用は安定化するどころか、品質や保守の課題に起因すると思われるトラブルが、依然として多く報告されています。

私たちは、ソフトウェア保守に関心を持つ方々に、広く SERC を知っていただいて、共に切磋琢磨することで、自らの成長を期待すると共に、この活動が、参加者の皆さんのモチベーション改善と広く実社会のソフトウェア保守業務の改善に貢献できることを願っています。そして、いつの日か、日本の誇る技術として、ソフトウェア保守技術が海外に輸出されることを夢見ています。

(株)アイ・ティ・フロンティア　田中　創

日本はもうすぐ4人に1人が65歳以上になる。

「人生65年時代が前提の仕組みは限界」。政府が6月に公表した高齢社会白書はこう警告する。シニアを従来通り「高齢者」と気遣い続ければ、社会は立ちゆかなくなるであろう。

そこで高齢者雇用専門会社を作り、事業開拓に挑んではどうか。

保守の現場では、保守のノウハウが伝承されていないことが想像できる。

これからの保守を考えた時、無人化の問題が保守の現場の生産性や品質に大きな影響を与えるのではと杞憂する。

ならば、定年退職した保守のプロフェショナルを起用し、そのノウハウを現役世代へ継承する事業が考えられる。

シニアの力を生かす仕組みを作らなけらば、これからの保守の問題を乗り切ることはできないのではないか。

東芝ソリューション(株) 増井 和也

今年 2 月 20 日、一般社団法人日本規格協会から 1996 年に発行された SLCP の JIS 規格の改訂版 JIS X0160:2012 ソフトウェアライフサイクルプロセス（原国際規格：ISO/IEC12207:2008）が公開出版されました。その解説には「～ ソフトウェア製品の供給，開発，運用，保守及び廃棄をするときに，適用。」と書かれていますが、規格自体を読むと、旧版にあった「開発プロセス」が無くなり、替わりに「実装プロセス」という名前になっています。

変更の背景は、ISO/IEC12207 として ISO/IEC15288（システムライフサイクルプロセスの国際規格）と調和を取るためのようです。背景がどうあれ、今回の改訂で「ソフトウェア開発」という言葉が使われなくなったことはソフトウェア保守にとって一歩前進だと私は考えています。

その理由は、「開発」の範囲の捕らえ方が曖昧であることが、ソフトウェア保守の範囲や保守プロセスの理解を妨げていたと私は考えていたからです。みなさんが「ソフトウェア開発」という言葉の使用をやめ「ソフトウェア実装」を使い、「実装」の範囲を正確に把握すれば、ソフトウェア保守の範囲や保守プロセスが自ずと正確に見えてくるだろうというのが私の見方です。

当研究会のみなさんには、ソフトウェアの保守や保守開発（改め保守実装？）の範囲やあるべきそれらのプロセスを考える際、今回の JIS X0160 の改訂版（2012 版）を参考にされることをお進めしたいと思います。

NARAコンサルティング 奈良 隆正

2 年前のこの報告書に、日本初の国産旅客機ＹＳ－１１が開発では成功したがビジネスでは失敗した、その原因は初めから保守を考えてい無かったことに有る、と紹介した。

今、ボーイング社の旅客機部門担当の社長が来日していて、話題のボーイング787についてNHKのインタビューに応えていた。皆さんご存知の通りボーイング787の機体の30％強は日本が製造を受け持っている。インタビュアーの「B787 の開発でこれからの課題は何か？」との問いに、社長は「燃費の向上と保守」と応えていたのが印象的であった。またこの解説では、国産機第 2 弾のMRJが保守に追従できるかがビジネスの課題だとも述べている。

燃費の向上は我々のソフトウェアに置き換えてみると、運用性の向上と捉える事が出来る。即ち、ISO/IEC-9216 でいう【運用時の品質】に相当すると考える。しかし運用における品質は未だに認知度が低いのが課題である。運用には保守が付きまとう理解も不十分なように思える。

また、航空機の保守は部品供給と保守情報のタイムリーな連絡である。これをソフトウェアに置き換えると、ISO1476(JIS X0161)の「ソフトウェア保守開発」そのもので有るように思える。しかしソフトウェア保守には依然として光が射して来ない。

このインタビュー記事を見て、航空機にしろソフトウェアにしろ、何れにしても保守がビジネスを制すると受けとめるる事が出来ると思うが、皆さんは如何がお考えだろうか。ソフトウェア保守は経費ベースでいうとライフサイクルの80％を占めていると言われその重要性が叫ばれてから久しいが、対応については依然として？マークが付く。経営層の方々には是非とも、ソフトウェア保守の重要性を真に理解して頂き、特に人材面での強化と、我々が提案している「二こぶラクダ型のコスト構造」に対応する手立てを切に御願いしたいと思う。

アイエックス・ナレッジ(株)　田中　一夫（事務局）

今年度も、前年度に引き続き、フォーラムを開催して、情報発信しようよと進めたが、中々難しいものです。結局 3 回しかできませんでした。来年こそは・・・と思うこのごろです。また、HomePage 更新も中々できず、会員の皆さんにはご迷惑をおかけいたした。

と昨年と同じ言葉を書いてしまった。困った事です。情報発信ならびに情報共有のために、HomePage も改訂し、SNS（OpenPNE）も導入しました。この SNS を有効活用すべく策をうちます。

4. 次年度募集案内

当研究会の活動期間は、11 月から 10 月です。現在、4つのグループが研究を行うべく計画を立てていますので、ここに紹介します。

＝＝

Aグループ研究テーマ 『保守技術者の教育』

開発者の教育あるいは教育コンテンツはあるのに、開発より難易度の高い保守に関しては、教育是非論を述べる人は多いが、実際の教育コンテンツや教育について検討しているのは少ない。当グループでは、その様な背景を踏まえ、現場に役立つ教育コンテンツと教育手法について、研究します。

SERCでは、第 4 年度・5 年度で教育コンテンツについて研究されていたが、ソフトウェア進化の中で、コンテンツの見直しも必要になっており、今回、2 年計画で保守技術者のための、教育方法と教育コンテンツを検討いたします。

＝＝

B グループ研究テーマ 『保守作業改善の基盤技術』

B グループでは、保守作業改善の基盤技術に関する調査研究を行なっています.
この研究成果を実際の保守・開発の実務に適用することにより, 保守開発エンジニアの能力を何倍にも強化(enhance), 増幅(enpower)し, メインテナンスの度に複雑化、巨大化したソフトウェアシステムの理解, 変更箇所の調査, テスト等を, より短期間により効率よく実施可能とすることを目指しています.

今年度の具体的な調査研究項目は以下の通りです.

- ソフトウェア・ツールの研究
  - ソフトウェア保守・進化に有用なツールの調査、評価
- ソフトウェア・サステナビリティの研究
  - ソフトウェア・サステナビリティの動向調査
  - ソフトウェア・サステナビリティの概念を把握する活動
- その他ソフトウェア進化・保守を支える考え方・技法の調査
  - ソフトウェア保守開発以外の分野に関して、ソフトウェア保守・進化と共通の性格を持つ、考え方や技法の調査、把握

その他, グループメンバーの興味の方向に応じた調査研究項目の設定も考慮に入れます.

これらの調査研究に関心がある保守開発関係者のみなさんの, 本グループへの参加を募ります.

＝＝

C グループ研究テーマ『10年後のソフトウェア保守を考える』

サブテーマ):誰が誰に何を伝承すべきなのか

内容)
・若手エンジニアの育成は、複雑化するソフトウェア保守環境において、重要度を増していくと思われる。
・次年度は「誰が誰に何を伝承すべきなのか」についての方法論やツールなどの研究を進めて行きたい。
・まずは、フタコブらくだの「問題の把握のよび修正分析」に必要なノウハウの伝承に取り組む予定です。

＝＝

D グループ研究テーマ『SERC が考える保守とは』

　当グループは「SERC が考える保守とは」をテーマに、研究会の内外に向けての情報発信・啓蒙活動を行っております。近年の主な活動には、ISO/IEC　14764（SoftwaeEngineering --Software Life Cycle Processes -- Maintenance）の JIS 化作業へのメバーの参画、解説書「ソフトウェア保守開発―ISO14764 による」の発行、ソフトウェアシンポジウム、ソフトウェア品質シンポジウムへの参加等があります。
　昨年度より、SERC 発足以来 20 年間に蓄積されたソフトウェア保守に関する知識を構造化し、ソフトウェア保守に関する概念と方法論に関わる知識体系をまとめあげること目標に活動を行っています。
　ソフトウェア保守に関する暗黙知を形式知化することにより、ソフトウェア保守担当者の育成に役立たせ、ソフトウェア保守に関する最新のテーマを整理体系化することによって、ソフトウェア保守技術の認知をはかり、ソフトウェア保守プロセスの確立を目指す組織の支援となることを目的としております。
ソフトウェア保守現場の改善を考えたい方や「ソフトウェア保守は〇〇だ」と義憤を感じている方も、皆さんの参加をお待ちしております。

＝＝

入会案内：http://serc-j.jp/serc/application/