



IBM Software Group

Rational Developer for System zのご紹介

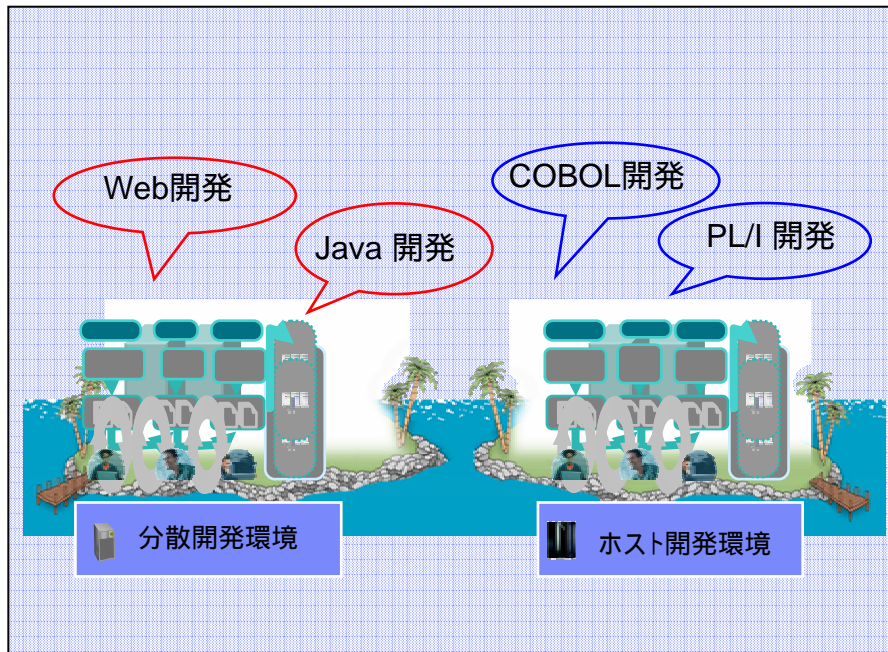
COBOL, PL/I統合開発環境

日本アイビーエム株式会社
ソフトウェア事業 ラショナル事業部

Rational software

プラットフォーム、ミドルウェアや言語に依存した開発環境

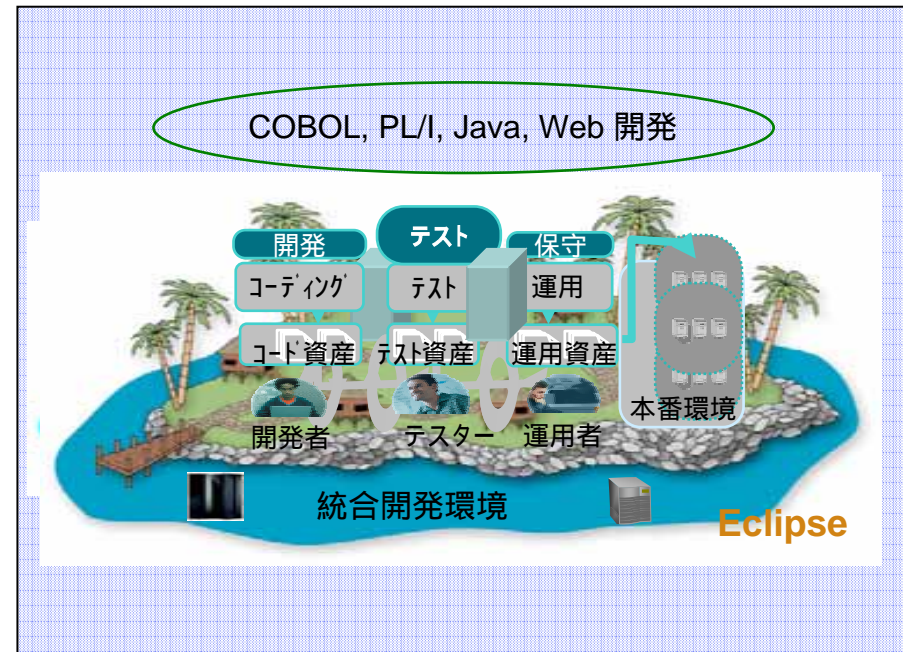
現状



現状の課題

- ✓アプリケーション毎に開発のプラットフォーム・ミドルウェア・言語が異なり、要員が個別に必要なになる。
- ✓開発ツールがプラットフォームにより異なり、ツールに関するスキルの横展開ができていない。
- ✓Javaの開発者は多くなっている反面、COBOLやPL/Iの開発者を確保するのが困難になっている。

今後



開発基盤の共通化による解決

- ✓開発基盤を共通化することにより、言語に依存しない共通のユーザー・インターフェイスで開発できます。
- ✓オープンソースであるEclipseのスキルがあれば、プログラム言語の習得だけで開発が可能です。
- ✓COBOL, PL/Iアプリの開発をホストに接続しなくても、PC環境にてオフラインで実施することができます。

アプリケーション開発基盤の共通化

コマンド・ドリブンのユーザー・インターフェイスであるTSO/ISPFではなく、グラフィカルなインターフェイスであるWindowsのEclipseの共通基盤上で、言語に依存しないインターフェイスを実現

- Eclipseでの画面 & 操作性統一 - COBOL & PL/IとJavaの共存
- 一貫した、分散システム開発とメインフレームでのテスト & 展開手法の確立
- 開発 & 運用メンバーの生産性向上

Rational Developer for System z (RDz)

< Eclipse 開発環境 >

ナビゲーター

コードの変更

エディター

選択して構文チェック、コンパイル、実行

問題

問題のエラーをダブルクリック

< 既存のTSO/ISPF開発環境 >

COBOLやPL/I開発者が減少する中、Java開発者からの移行を容易に可能にします。またオフショア開発などの事例等も増しています。

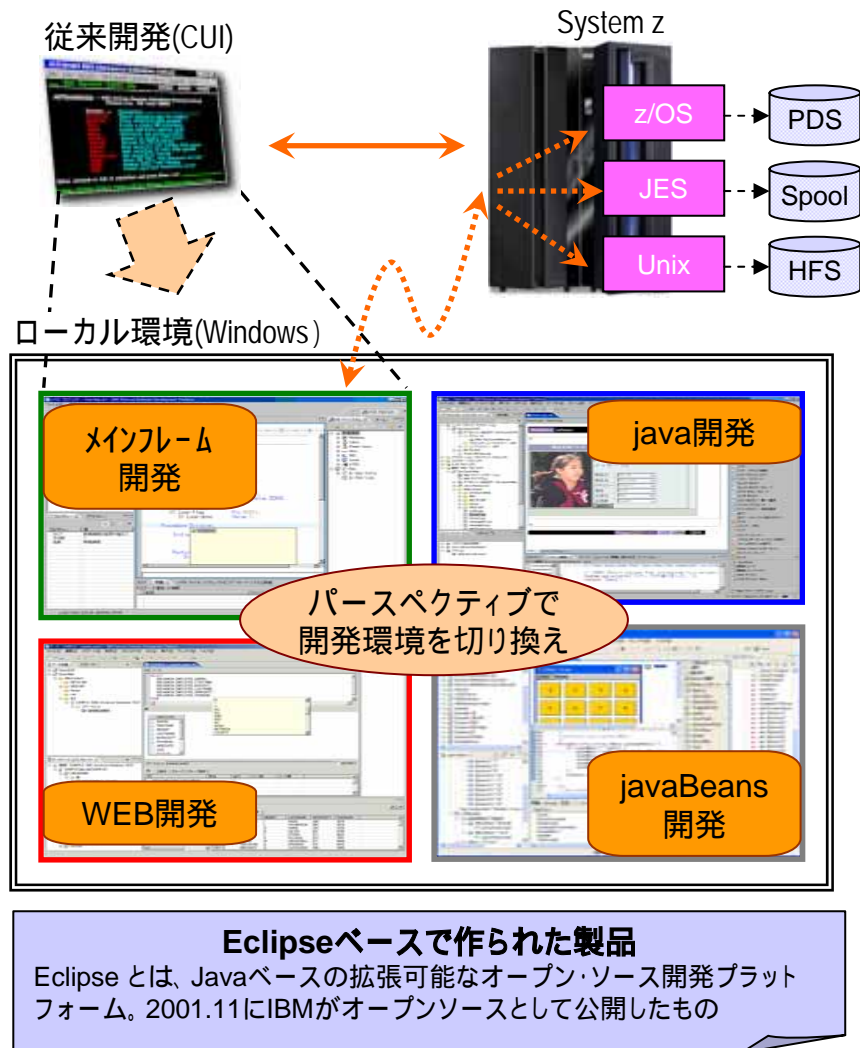
- Windows上で、コード編集、コンパイル&リンク、実行、およびデバッグが可能
- メインフレームへのリモート接続による、ライブラリー編集、JCLサブミット、実行、ジョブのモニター、分散デバッグなどが可能

Rational Developer for System z (RDz) とは・・・

分散環境の開発に加えてz/OS環境(COBOL, PL/I, アセンブラ)の開発もWindows環境で行える開発ツールです。

RDz の主な特徴

1. Eclipseベースのマルチ統合開発環境 (IDE)
開発者は自分に必要な開発に応じて開発環境を選択
追加したい機能はプラグインで容易に追加可能
2. Rational Application Developerの機能を拡張
RADのJava、C++基本開発機能に加え、
ホストCOBOL、PL/Iコンパイラを含む開発機能を同梱
3. 豊富な入力支援機能
言語に応じた構文検査、コンテンツ・アシスト、世代管理、
JCL自動生成、マニュアル連携など機能が充実
4. リモート開発機能
ホストのプログラム・ソース(PDS)に直接アクセスしたりJCLの
実行が可能。
別製品であるデバッグツールやファイルマネージャーとの連
携も可能。(後述)
5. SOA化支援機能
COBOLアプリケーションをコンポーネント化
6. チーム開発
Rational製品であるClearCase/ClearQuestと連携し
チームによるシームレスな並行開発を実現。



統合開発環境を実現するRational製品

WebSphere Studio
Development Suite for HLL/WB

- 日本語仕様書からCOBOLやPL/Iの生成
- ソースから日本語仕様書へリバース
- 用語辞書管理

Rational Developer for System z

- COBOLとPL/I: ローカルやリモートで編集、コンパイル、デバッグ
- COBOLのXML活用
- EGL (COBOL)

Rational Application Developer

- Webアプリケーション開発環境
- Strutsビルダー
- Webサイト・デザイナー
- JSF/WDO
- Webサービス
- XML開発環境
- EJB開発環境
- ポータルツールキット

Appl設計者

Software Architect

Web Appl開発者

J2EE Appl開発者

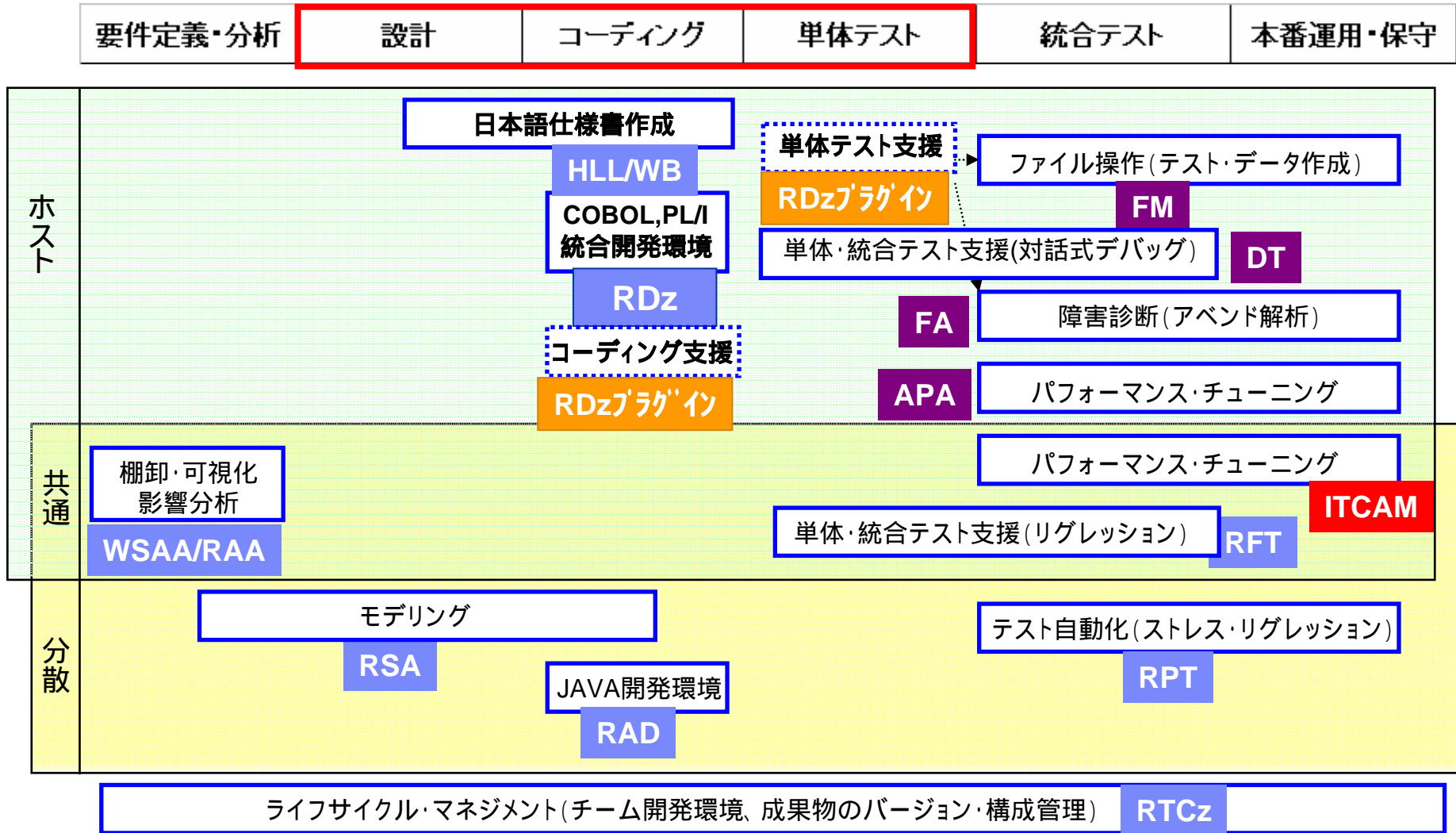
z/OS Appl開発者



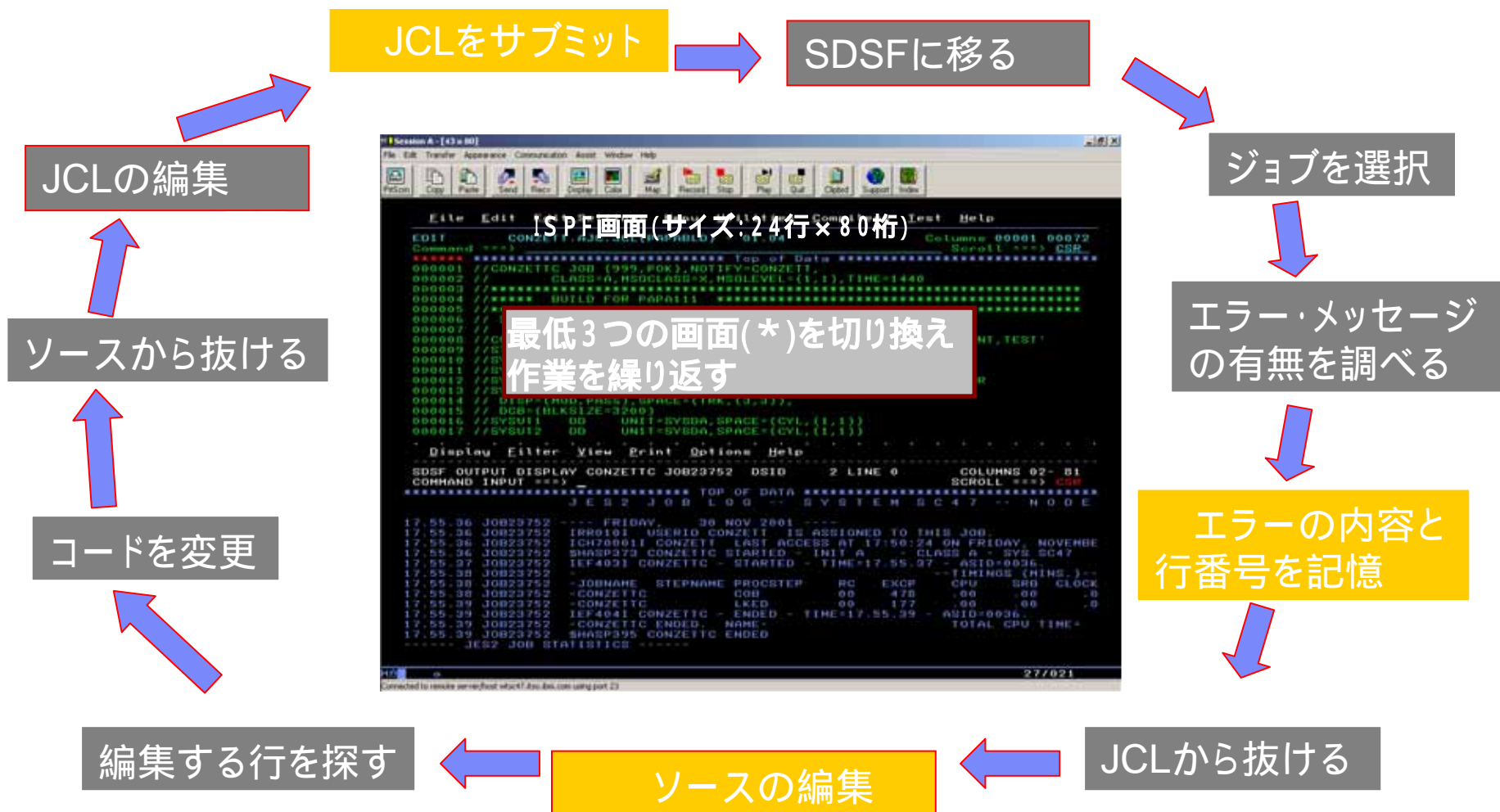
(オープンソース) Java編集, コンパイル, デバッグ



Rational Developer for System z(RDz)の位置付け



従来の開発作業(ISPF)



RDzを使った開発作業

The screenshot displays the IBM WebSphere Developer for System z interface. The main window shows a COBOL program named GETEMP.cbl with the following code:

```

行 46 列 82 挿入
-----#A-1-B-----2-----3-----4-----5-----6
      ** WORKING STORAGE STARTS HERE** *.
77  KEYNUM          PIC 9(8).
77  COMLEN         PIC 99(11) COMP.
77  RESPONSE       PIC X(80).
      *
      FILE.
02  FILEREC.
03  STAT          PIC X.
03  NUMB          PIC X(8).
03  NAME          PIC X(20).
03  ADDR          PIC X(20).
03  PHONE         PIC X(8).
03  DATEX         PIC X(8).
03  AMOUNT        PIC X(8).
03  COMMENT       PIC X(8).
01  COMMAREA      PIC X(80).
02  FILEREC.
03  STAT          PIC X.
03  NUMB          PIC X(8).
03  NAME          PIC X(20).
03  ADDR          PIC X(20).
03  PHONE         PIC X(8).
03  DATEX         PIC X(8).
03  AMOUNT        PIC X(8).
03  COMMENT       PIC X(8).
      *
      LINKAGE SECTION.
      *
      EXEC CICS HANDLE CONDITION ERROR(ERRORS) END-EXEC.
      *
      MOVE SPACES TO FILE.
      MOVE
      MOVE NUMB IN COMMAREA TO KEYNUM.
      *
      EXEC CICS READ FILE(FILE) INTO(FILE)
      RESP(RESPONSE) RIDFLD(KEYNUM) END-EXEC.
      *
00180000
00190000
00200000
00210000
00220000
00230000
00240000
00250000
00260000
00270000
00280000
00290000
00300000
00310000
00320000
00330000
00340000
00350000
00360000
00370000
00380000
00390000
00400000
00410000
00420000
00430000
00440000
00450000
00460000
00470000
00480000
00490000
00500000
00510000
00520000
    
```

Annotations and callouts in the image include:

- ナビゲーター** (Navigator): Points to the left-hand pane showing a project tree.
- エディター** (Editor): Points to the main code editing area.
- ソースの編集** (Source Editing): A yellow box highlighting the code.
- コンパイラを使った構文検査の実行** (Execution of syntax check using compiler): A yellow box with an arrow pointing to the compiler options pane.
- 画面の切り換えは不要 可視領域が広く見やすい エラー箇所を探さなくて良い** (No need to switch screens, visible area is wide and easy to see, no need to search for error locations): A blue box with white text.
- エラーメッセージ行 [X] をダブルクリックでエラー箇所にジャンプ** (Jump to error location by double-clicking error message line [X]): A yellow box with an arrow pointing to an error message in the bottom pane.
- メッセージ欄** (Message pane): A yellow box pointing to the bottom pane showing error details.

The error message in the bottom pane is:

ID	メッセージ	重..	行	ロケーション	ホスト名	日付
IGYPS2106	IGYPS2106-S "MOVE" ステートメントで "MOV...	2	46	/TESTEMP/cobol/G...	Local	2007/08/01 3:00:19

RDzのローカル・コンパイラを使った開発の流れ

** ローカル環境(Windows上) **

プログラム・ソースの作成

* コンテンツ・アシストなど豊富な入力支援機能を活用。

プログラムの**構文検査**・・・ RDzの標準機能では通常ここまで

* ローカル・コンパイラを使ってコンパイル・エラーを除去

ロードモジュールの作成

！ テストデータの準備と環境構築が必要

* ローカル・コンパイラを使ってモジュール生成

プログラムの単体テスト

* CICS for Windows(Txseries), DB2 for Windowsを利用可能

プログラムのデバッグ

* ローカル・デバッグ機能を使用してSTEP by STEPでモジュール実行

プログラム・ソースをリモート環境(zServer)に転送

* PCフォルダからzServer上のPDSへソースを転送(ドラッグ&ドロップ)

** リモート環境(zServer上) ** (RDz インターフェイスから作業)

ロードモジュールの作成

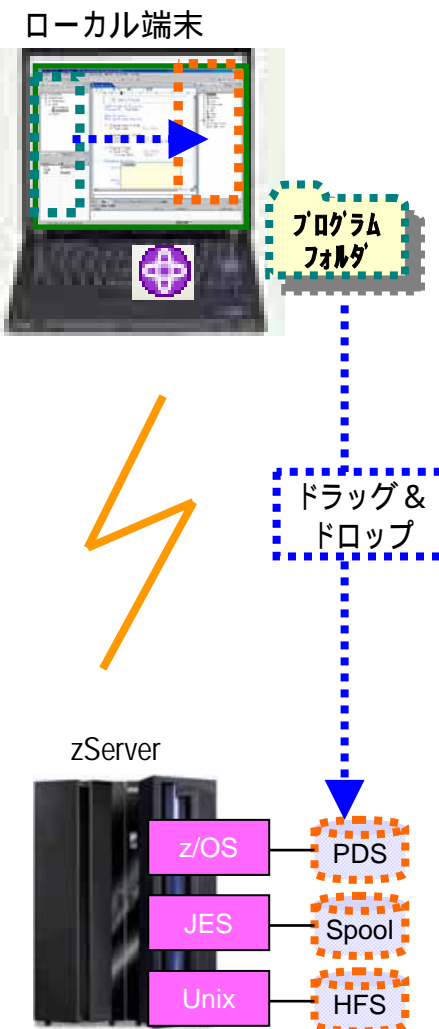
* JCL自動生成機能を使用してモジュール生成

プログラムの単体テスト

* テストJCLをRDz インターフェイスよりジョブ実行

プログラムのデバッグ

* Debugging TOOLと連携しRDz上でデバッグを実施



エミュレータ機能: 開発作業以外のオペレーションが必要なケース

弊社Personal Communicationと機能的にほぼ同等であり、複数ウィンドウを起動可能。
 ファンクション(PF)キーはもちろん、ボタン形式のPFキーも用意されている。
 (Enterキーは『実行』扱い。『Enter』は、Shift+Enter)

```

Display Filter View Print Options Help
-----
SDSF SYSLOG 1298.101 SYS1 SYS1 09/09/2007 0W 2987 COLUMNS 1 80
COMMAND INPUT ==> SCROLL ==> CSR
N 4000000 CPAC 07252 20:41:27.36 TSU01320 00000090 $HASP373 IBMUSER STARTE
N 0000000 CPAC 07252 20:41:27.54 TSU01320 00000090 IEF1251 IBMUSER - LOGGED
N 0000000 CPAC 07252 20:41:38.85 00000281 IEF1981 IEF237I 0100 ALL
N 0000000 CPAC 07252 20:41:38.89 00000281 IEF1981 IEF285I SYS1.C
N 0000000 CPAC 07252 20:41:38.90 00000281 IEF1981 IEF285I VOL SE
N 0000000 CPAC 07252 20:42:30.03 TSU01320 00000290 IEA830I OPERATOR IBMUSE
NC0000000 CPAC 07252 20:42:30.19 IBMUSER 00000290 D A.L

Menu Options View Utilities Compilers Help
-----
DSLIST - Data Sets Matching MASA.DEMO* Row 1 of 8
Command ==> Scroll ==> PAGE

Command - Enter "/" to select action Message Volume
-----
MASA.DEMO.COBOL SWGAD1
MASA.DEMO.COPYLIB SWGAD1
MASA.DEMO.JCL SWGAD1
MASA.DEMO.LOAD SWGAD1
MASA.DEMO.OBJ SWGAD1
  
```

アプリケーションのコンポーネント化を支援 (SOA対応への支援機能)

CICS、IMSアプリのコンポーネント化を支援。ブラウザや .NETなどから呼び出す事が出来る。

例) CICSアプリケーションのSOA対応の流れ

STEP0.コンポーネント化対象プログラムの洗い出し



SEND/
RECEIVE

STEP1. PLとBL部分でモジュール分割
(BL部分のコンポーネント化)



SEND/
RECEIVE

STEP2. COMMAREA情報を元にWSDLと
アダプタ・コードを生成

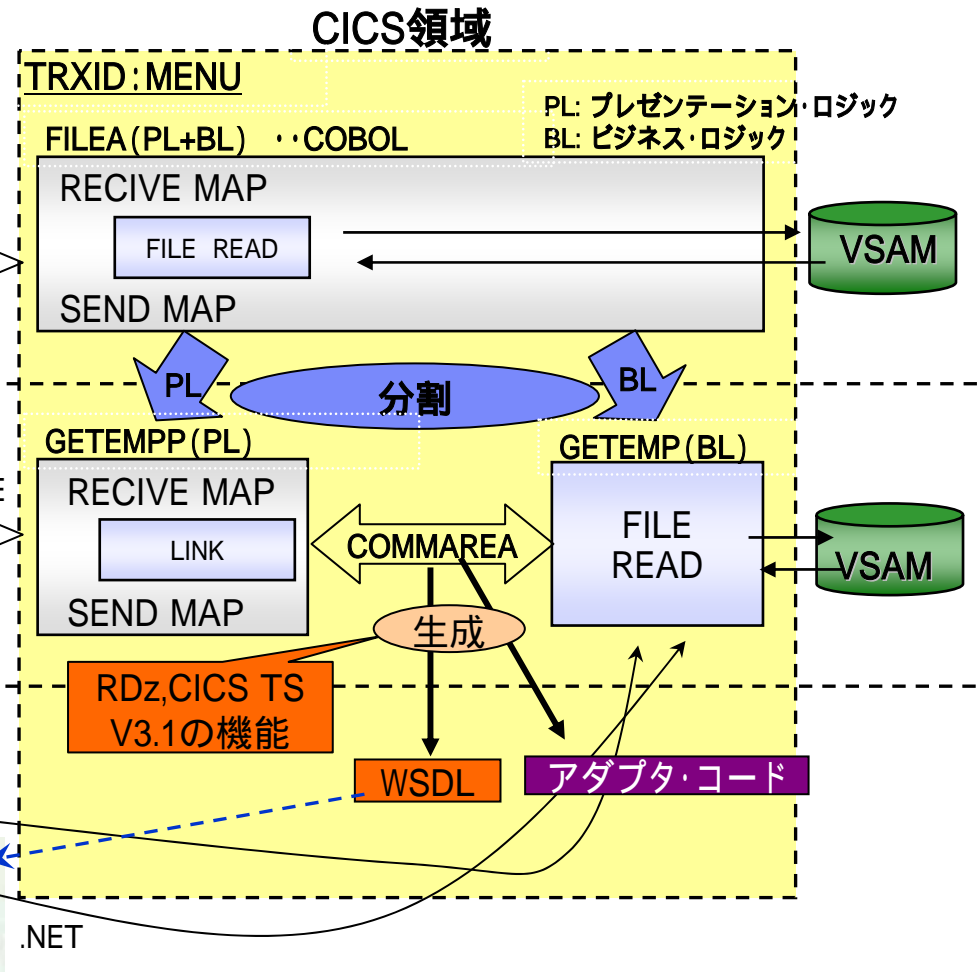


ブラウザ

STEP3. WSDLをベースにアプリを作成。
アダプタ・コード経由で接続する。

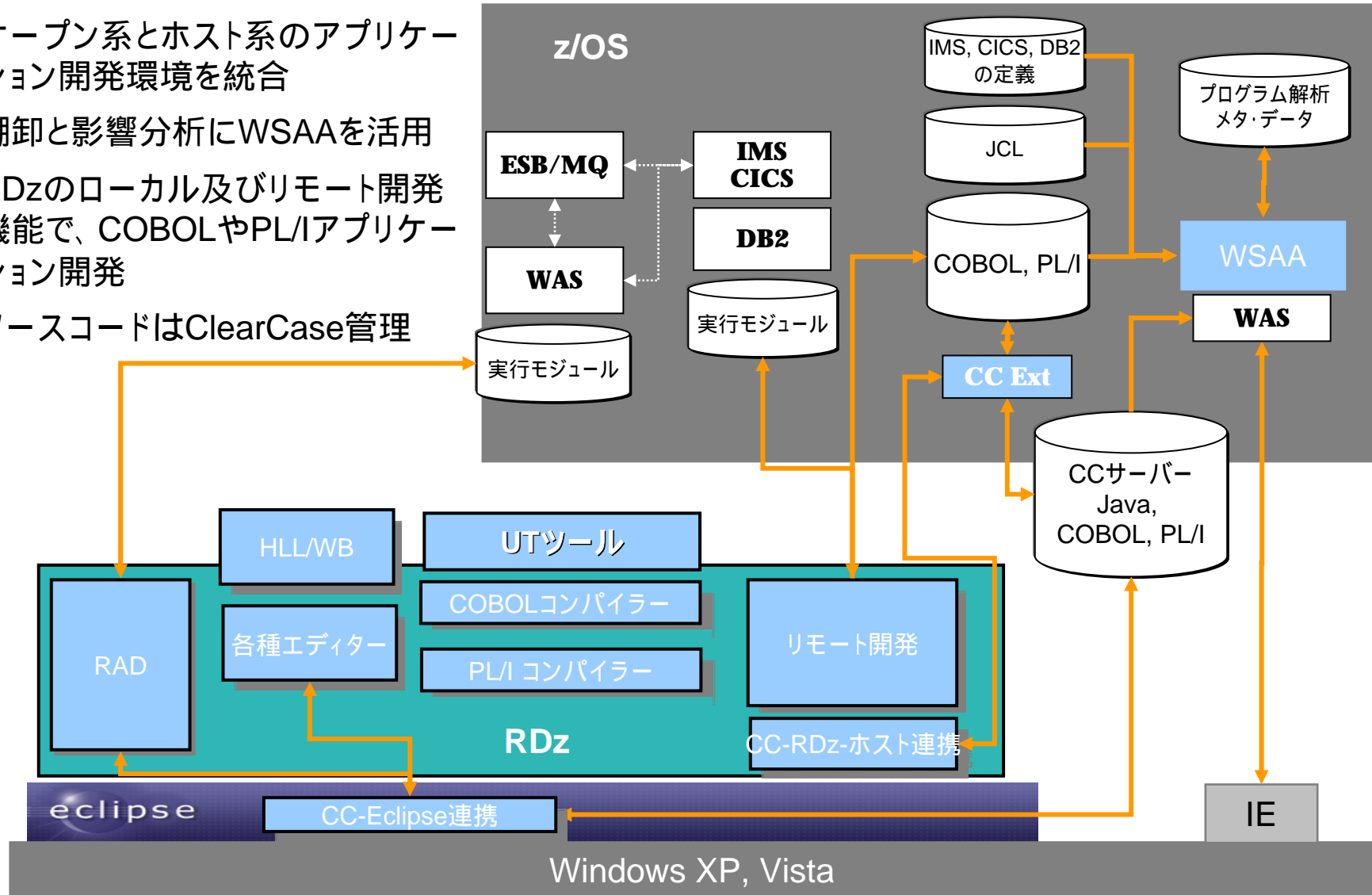


.NET



アプリケーション統合開発環境事例

- オープン系とホスト系のアプリケーション開発環境を統合
- 棚卸と影響分析にWSAAを活用
- RDzのローカル及びリモート開発機能で、COBOLやPL/Iアプリケーション開発
- ソースコードはClearCase管理



RDz 採用による効果

- **コスト削減**
 - オフライン開発によるCPUコストの削減
 - 生産性向上による開発費用の削減

- **スキル育成・継承、横展開が可能**
 - 豊富な開発支援機能によりホストスキルがなくても開発が可能
 - JCL自動生成機能
 - SPOOL、データセットのフォルダイメージでの表示
 - Java開発者によるCOBOL開発、COBOL開発者によるJava開発
 - Javaの開発ツールの主流はEclipseであり、同じインターフェースでCOBOL開発が可能
 - 開発者が新たに1つ言語を習得することは困難ではなく、インターフェースの違いに戸惑う

- **新時代への適応**
 - 既存アプリケーションのサービス化支援機能により、少ないワークロードで既存資産を有効活用可能
 - EGL採用により、新規アプリケーション開発、新しいニーズへの早期対応が可能



(ご参考)RDz生産性向上によるコスト削減効果 (日本での実績)

* プログラム編集とコンパイルエラー除去(プログラムが完成するまでのトータル時間)

開発作業項目	従来(分)	RDz(分)	備考(RDzとの違い・手順)
①プログラム編集 ※生産性			
1. ISPFエディタでプログラム・ソース編集 ・24×80画面での編集 ・行コマンドでコピー・ペースト・削除 ・1画面で同じ色 ・変数エリアの参照。画面をスクロールし定義域を確認	新規開発(500step)	1440.0 *3日	1. LPEXエディタでプログラム・ソース編集 ・解像度が許す限りの行・桁数を表示。複数並べ表示可能。 ・PF(←CtrlShift)によるショートカットキー。*ブロックコピー・行コマンドの利用も可能。 ・コメント、関数、エリア、エリア属性など色分けにより見やすく表示 ・コンテンツアシスト機能により、次の入力候補となる変数やパラメータを一覧表示。 ・アウトライン機能による全体フロートの表示とクリックにより該当箇所へのジャンプが可能。 2. マニュアル参照(1回) ・PDFもしくは紙ベースで参照。該当箇所をたどり着くまで。
	変更作業(500step)	240.0 *0.5日	
②コンパイルエラーの除去 ※下記は実績値			
1. コンパイル実行と結果参照 ・JCLのデークセットをオープン ・JCLの編集・内容確認(カブカ) ・JCLの実行 ・SDSF画面への切り換え ・JOB LOGの検索、オープン 2. SDSF画面で1つのエラーあたりの内容把握(エラーの数だけ繰り返し) ・最下段のメッセージ一覧にてエラーメッセージと行番号を確認 ・エラー箇所の発見と前後の確認(24×80のため数回スクロール) ・エラー内容の把握 3. ソース編集画面でエラー箇所を確認(エラーの数だけ繰り返し) ※SDSFからPF9で画面切換えし該当箇所をサーチ	3.0	0.5	2. エラーメッセージ1つあたりの内容把握(エラーの数だけ繰り返し) ・画面下段のメッセージ欄に表示されているメッセージをダブルクリック(該当行へジャンプする) ・エラー内容の把握 3. エラー箇所の確認(エラーの数だけ繰り返し) ※上記2. の作業に含まれるため、操作はなし
	3.0	0.5	
	0.1	0.0	

① 赤点線内の生産性値をお客様のご経験値で埋めてください。この値は、経験値です。値は環境、開発者のスキルにより変わります。

◆COBOL PGM1本の **新規作成生産性(500ステップ相当)** 想定回数 従来(分) RDz(分)

①エディタ編集	1440.0	1152.0
マニュアル参照回数	15	3.0
②コンパイル回数(エラーがなくなるまでの延べ回数)	20	10.0
エラー個数	50	25.0
エラー箇所のサーチ	50	0.0
小計	1670.0	1190.0

② 上記を踏まえて、開発者が1年間で新規開発する際の実績を従来形式と、RDz採用ケースで計算。

◆COBOL PGM1本の **メンテナンス生産性(同上)** 想定回数 従来(分) RDz(分)

①エディタ編集	240.0	192.0
マニュアル参照回数	5	1.0
②コンパイル回数(エラーがなくなるまでの延べ回数)	10	5.0
エラー個数	25	12.5
エラー箇所のサーチ	25	0.0
小計	352.5	210.5

③ 同じく、開発者が1年間でメンテナンスする際の実績を従来形式と、RDz採用ケースで計算。想定回数を記入。

◆年間開発本数による工数算出(①+②部分)(1人あたり) 想定本数 従来(分) RDz(分)

新規COBOLプログラム作成	5	8350.0	5950.0
COBOLメンテナンス	20	7050.0	4210.0
計(分)		15400.0	10160.0
計(日)		32.1	21.2
計(月)		1.6	1.1
計(万円)		133.7	88.2

④ 年間の新規開発とメンテナンスの想定本数を記入。米国ビジネスケースでは1:4の割合

⑤ 一人当たりの年間の削減効果が算出される。

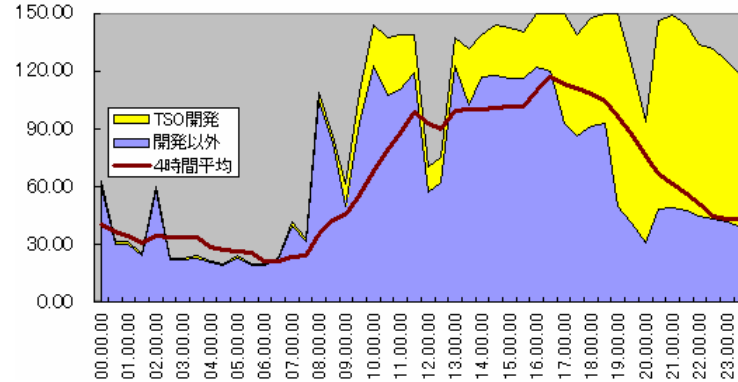
削減額(万円) 45.5 *年間開発コスト削減(開発者単価:1000万/年を想定)

(ご参考) RDzオンライン開発によるCPUコスト削減効果

CPU使用率とTSOコストの推移(30分毎)

項目種類	基礎情報		TSOコストのうち100%が コンパイルUp作業		TSOコストのうち65%が コンパイルUp作業				
	日	時	SYS2 使用 MIPS	TSO 使用 MIPS	削減後 SYS2 MIPS(2)	削減後 4Hour Ave(2)*1	削減後 MIPS ②*65%	削減後 SYS2 MIPS(1)	削減後 4Hour Ave(1)*1
9月5日	00:00:00		62.50	2.50	60.00	40.20	1.63	60.88	45.22
	00:30:00		31.25	1.25	30.00	36.41	0.81	30.44	40.34
	01:00:00		31.25	1.25	30.00	34.36	0.81	30.44	37.48
	01:30:00		25.00	1.00	24.00	30.58	0.65	24.35	32.73
	02:00:00		60.00	2.40	57.60	34.17	1.56	58.44	35.88
	02:30:00		22.50	0.90	21.60	33.81	0.59	21.92	35.10
	03:00:00		22.50	0.90	21.60	33.44	0.59	21.92	34.32
	03:30:00		23.75	0.95	22.80	33.45	0.62	23.13	33.94
	04:00:00		21.25	0.85	20.40	28.50	0.55	20.70	28.92
	04:30:00		20.00	0.80	19.20	27.15	0.52	19.48	27.55
	05:00:00		23.75	0.95	22.80	26.25	0.62	23.13	26.63
	05:30:00		20.00	0.80	19.20	25.65	0.52	19.48	26.02
	06:00:00		20.00	0.80	19.20	20.85	0.52	19.48	21.15
	06:30:00		22.50	0.90	21.60	20.95	0.59	21.92	21.15
	07:00:00		41.25	1.65	39.60	23.10	1.07	40.18	23.44
	07:30:00		32.50	1.30	31.20	24.15	0.85	31.66	24.50
	08:00:00		108.75	4.35	104.40	34.65	2.83	105.92	35.16
	08:30:00		86.25	3.45	82.80	42.60	2.24	84.01	43.22
	09:00:00		61.25	11.67	49.58	45.95	7.58	53.67	47.04
	09:30:00		108.75	16.25	92.50	55.11	10.56	98.19	56.88
	10:00:00		143.75	21.25	122.50	68.02	13.81	129.94	70.68
	10:30:00		137.50	30.00	107.50	78.76	19.50	118.00	82.69
	11:00:00		138.75	28.75	110.00	87.56	18.69	120.06	92.68
	11:30:00		138.75	20.00	118.75	98.50	13.00	125.75	104.44
	12:00:00		70.00	12.92	57.08	92.59	8.40	61.60	98.90
	12:30:00		75.00	12.92	62.08	90.00	8.40	66.60	96.73
	13:00:00		137.50	15.00	122.50	99.11	9.75	127.75	105.99
	13:30:00		131.25	29.17	102.08	100.31	18.96	112.29	107.75
	14:00:00		138.75	21.67	117.08	99.64	14.08	124.67	107.09
	14:30:00		143.75	25.83	117.92	100.94	16.79	126.96	108.21
	15:00:00		142.50	26.67	115.83	101.67	17.33	125.17	108.85
	15:30:00		140.00	23.75	116.25	101.35	15.44	124.56	108.70
	16:00:00		150.00	27.92	122.08	109.48	18.15	131.85	117.48
	16:30:00		150.00	30.00	120.00	116.72	19.50	130.50	125.47
	17:00:00		150.00	57.00	93.00	113.03	37.05	112.95	123.62
	17:30:00		138.75	52.73	86.03	111.02	34.27	104.48	122.64
	18:00:00		147.50	56.05	91.45	107.82	36.43	111.07	120.94
	18:30:00		150.00	57.00	93.00	104.71	37.05	112.95	119.19
	19:00:00		150.00	100.50	49.50	96.41	65.33	84.68	114.13
	19:30:00		123.75	82.91	40.84	86.99	53.89	69.86	107.29
	20:00:00		93.75	62.81	30.94	75.59	40.83	52.92	97.43
	20:30:00		146.25	97.99	48.26	66.63	63.69	82.56	91.43
	21:00:00		148.75	99.66	49.09	61.14	64.78	83.97	87.81
	21:30:00		143.75	96.31	47.44	56.31	62.60	81.15	84.89
	22:00:00		133.75	89.61	44.14	50.40	58.25	75.50	80.45
	22:30:00		131.25	87.94	43.31	44.19	57.16	74.09	75.59
	23:00:00		125.00	83.75	41.25	43.16	54.44	70.56	73.83
	23:30:00		117.50	78.73	38.78	42.90	51.17	66.33	73.39

ケース1) TSO作業の100%が編集作業(*2)と仮定すると、上限は122.08Mipsまで下がる。



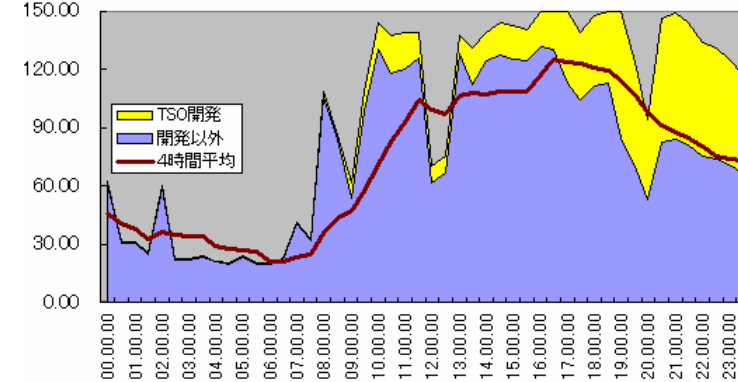
122.08Mips
116.72Mips(*)

WLC採用時は更に6Mips程度削減でき月18万,年216万程度の追加削減が期待できる。請求はマシンでの最大使用量

*1Mips= ¥ 1000/日と仮定

削減効果/月 : (150-122)Mips × 1000円 × 30日 = 840,000円 (84.0万円)
削減効果/年 : 削減効果/月 × 12ヶ月 = 10,800,000円 (1080万円)

ケース2) TSO作業の65%が編集作業(*2)と仮定すると、上限は131.85Mipsまで下がる。



131.85Mips
125.47Mips(*)

WLC採用時は更に6Mips程度削減でき月18万,年216万程度の追加削減が期待できる。請求はマシンでの最大使用量

*1Mips= ¥ 1000/日と仮定

削減効果/月 : (150-132)Mips × 1000円 × 30日 = 540,000円 (54.0万円)
削減効果/年 : 削減効果/月 × 12ヶ月 = 6,480,000円 (648万円)

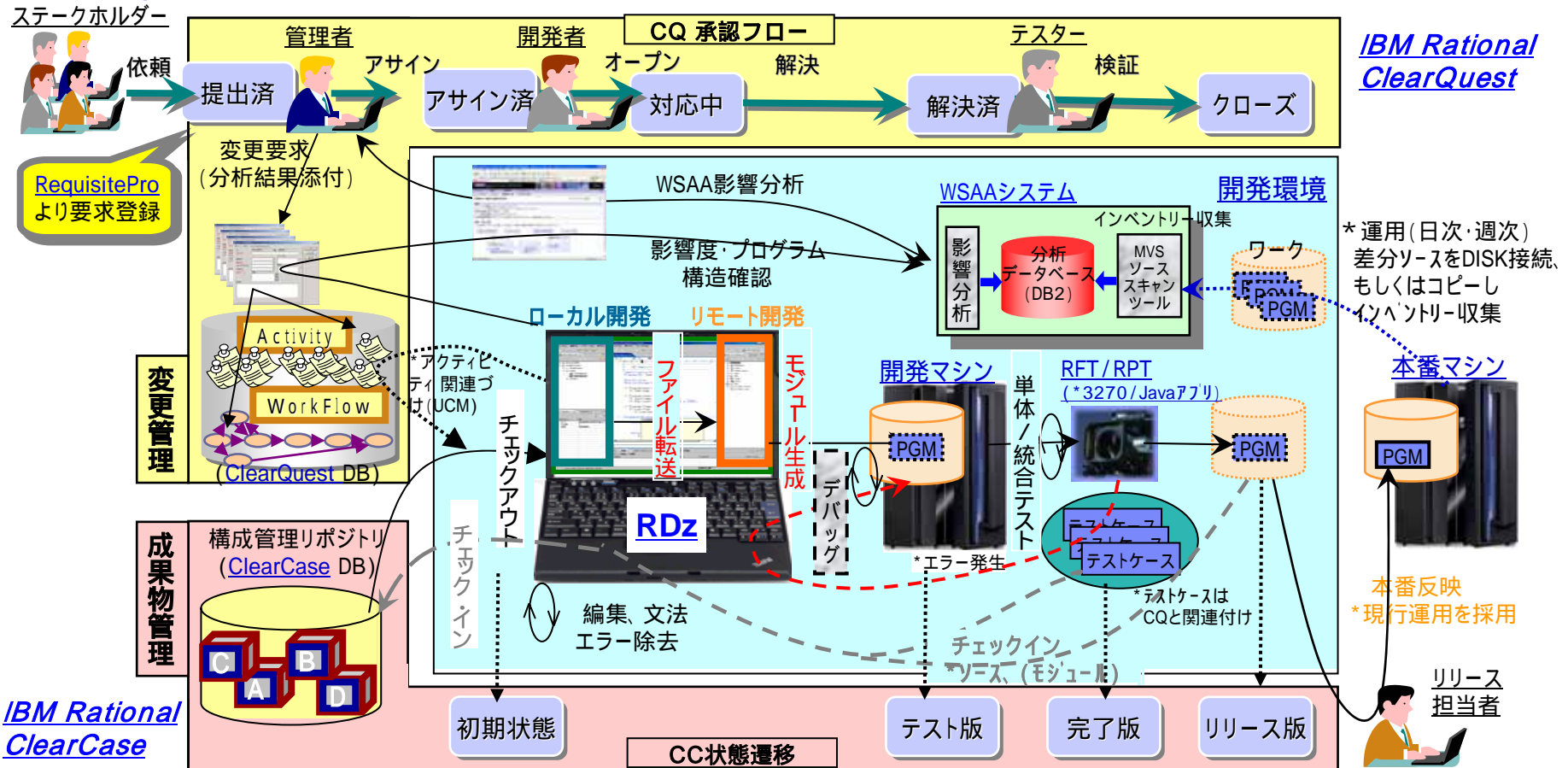
(*1) WLC契約時は4時間平均(4hour Ave)の最大値が課金となります。CPUはHW能力上限まで使用可能であり、SWキャッピング設定により費用を抑えられます。

(*2) 編集作業とは、オフライン開発可能な作業を指します。製品単体機能では、コンパイルエラーを取り除くまでの作業をパソコン上で行えます。

RDzはEclipseベースの統合開発ツールのため、プラグイン(YSL開発)でテスト機能を追加することにより、更にCPUコストの削減につながります



(ご参考) プログラム変更要求におけるRational製品の連携イメージ



処理フロー例

ステークホルダーから機能改善要求をRequisiteProに登録。
 管理者は関係者のアサインとWSAAを使い変更による影響分析を実施
 WSAの影響分析結果を確認し、CQに担当者別アクティビティを登録
 開発担当者はCQからWSAA分析結果とプログラム構造を参照する
 開発担当者はCQの変更要求アクティビティに従って
 該当のモジュールをCCよりチェックアウト
 ローカル・コンパイラを使用してコンパイル(文法)エラーを除去

RDz画面よりソースをドラッグ&ドロップでメインフレームにファイル転送
 RDzのJCL自動生成機能によりソースをコンパイル・リンク。ロードモジュールを作成
 テスト担当者はRFT,RPTのテストケースを再利用・追加編集してテストを実施
 * バッチなどはRDzのリモート接続画面からテストJCL実行し結果を参照する。
 エラーが発生した場合、RDz画面からエラー箇所をデバッグ(ホストのDebugToolと連携)
 テスト完了しソース、(モジュール)をRDz画面よりCCにチェック・イン
 リリース担当者が本番にモジュールとソース(WSA用一時ファイルとして)を反映
 さらにBuildForgeの採用により、モジュールのビルド、テスト、本番リリースの自動化が可能