12

Appendix 1- RDz Productivity Benchmarks

This Appendix documents a Benchmark completed in March 2010 that compared two IBM products: RDz v7.6.1 and ISPF v6.0.

The work performed in the study was a set of tasks normally associated with traditional z/OS COBOL maintenance and production support:

- Code analysis - paragraph flow and program structure analysis
- Navigating and modifying existing COBOL programs
- Adding a small amount of new business logic to an existing program
- Updating fields in copybooks and modifying code that referenced to updated fields
- Doing tradition Data Flow analysis
- Working with SQL and DB2 test data
- Compiling and linking a program

The detailed scripts used by participants in the study may be found on the ITSO server at: (Chris to provide)

12.1 Reasons for the study

Back in the early 90's there was a noticeable shift in the breeze within trendy, I/T magazines and academia - where writers argued that production workloads could be "downsized" off mainframes to distributed platforms. Outrageous and almost entirely unsubstantiated claims were made - which unfortunately went under-analzyed and the net was that entire web-sites now exist that document the cost and other unintended consequences of wishful thinking posing as research: http://www.actscorp.com/reboothill.htm

Even today, the idea that you can replace a mature, stable, scalable, powerful, manage-able centralized hardware platform with immature, (relatively) unstable, un-manage and un-supportable server farms and complex application stacks simply doesn't account for the complexity, scale and scope of "production workload" file and database I/O sizing, transaction and batch window throughput.

We're sorry - but except for very small shops or for small departmental applications, the most cost-effective place to run mission-critical and performance-snsitive enterprise applications is - and will be for a long, long time to come on a centralized z/Server.

However, **mainframe software development** done off the mainframe has been a beguiling concept for at least a few decades. Development would seem both practical and productive, given today's desktop technology. And of course, mainframe software development and maintenance is both practical and do-able with the tools of the optimized lifecycle, but... is it really more productive? What tangible evidence exists that proves RDz is actually a more productive development platform than green screen (ISPF)? And if so - exactly how much more productive? And for whom is it more productive.

These are questions that IBM management put to the Redbook team late in 2009. Specifically, they asked us to design a comparison study between two IDEs:

- ISPF version 6.0 running on a z/10
- RDz v7.6.1 running on thnk

The study was to be conducted as follows:

- An "apples to apples" comparison of the two IDEs
- A sizeable number of participants
- Focus on z/OS traditional maintenance tasks that are common, everyday
- A true research project - with double blind controls, etc.

12.2 Design of the Study

In attempting to satisfy the above criteria, the IBM team started by polling a number of ISPF/COBOL programmers to determine what kinds of activities were consistent day-in/day-out. The ISPF/COBOL programmers were from within IBM, from business partners and customers.

A large task list of 100 discrete activities was created for ISPF - and verified with the ISPF/COBOL programmer team. These 100 tasks were broken out into seven categories:

1. Code navigation
2. Edit operations (basic ISPF edit operations)
3. COBOL coding - adding a new paragraph to an existing COBOL program with changes to the copybooks
4. Data flow analysis - following the value in a variable as it is moved from f
5. Code/syntax error removal
6. Build - compile and link
7. DB2 and SQL work - modifying DB2 table values, creating new rows, writing and testing SQL statements

While this is obviously not an all-inclusive list of what z/OS developers do (not included are things such as: VSAM and data file maintenance, and debugging), after vetting the list with the ISPF/COBOL programmer group, it was determined that all of the tasks were standard fare, for developers week-in/week-out. However, we encourage you to pull down copies of the scripts and see for yourself (Chris... need the ITSO Server URL here- thanks)

12.3 Human Factors Avoidance

As far as possible, every attempt was made to remove "Human Factors" and bias from the research - by doing the following:

- As you will see from the scripts (figures 12-1 and 12-2), close-ended, click-for-click detailed instructions were follwowed to minimize:
  - Think time
  - Differences in
    - Product (ISPF or RDz) experience
    - Typing speed
    - Application development experience
- Project participants were told that they were trying to find gaps between RDz and ISPF functionality - this was a single-blind research tactic, attempting to mitigate subliminal bias
- 50% of those participants that did both the RDz and ISPF scripts did the RDz scripts first, and the other 1/2 did the ISPF script first. This was done to mitigate "learning and retention" bias

12.3.1 ISPF “Bias”

Speaking of bias, there were two areas of the study that were biased towards ISPF and against RDz:

1. The scripts were written from ISPF not an RDz perspective. In other words, we created the scripts based on what ISPF developers do daily with ISPF - and adjusted RDz’s script to functionally match what was done with ISPF. This biased the results against what might have been achieved had we started from an RDz perspective, and tried to match ISPF to the product’s capabilities.
2. The way the tests were administered was optimized for ISPF - not RDz. Because the scripts were exceedingly long (the 100 steps for ISPF were documented in 676 rows of a spreadsheet (at 10 pt font) - it was decided that printing off hard-copy and reading would not work. So instead, the participants were told to display the script alongside ISPF or RDz, and to “read and scroll”. This worked well for ISPF - as most participants were in 32/80 mode, and ISPF fit perfectly in view full-screen, but RDz had to be minimized, excessive horizontal scrolling was necessary and many of the benefits of RDz’s/Eclipse MS-Windows orientation were vastly reduced, because developers were viewing 2/3rds of the product.

12.3.2 Participant Characteristics

There were 23 participants in the study - from:

- Business partners
- Customers
- Academia
- IBM internal
  - ISPF consultants - mostly Global Business Services professionals
  - Rational tech-field

The average years of experience of participants were:

- ISPF - 12.7 years
- RDz - 1.3 years

Summary

Given the above control mechanisms it should be clear that every attempt was made to meet the research goals set down by IBM management (“apples to apples” comparison, etc.). For the most part, the standard deviation statistics bore out that we met these goals.

That said, this does not mean that the Benchmark results should be interpreted as academic (Underwriter’s Laboraties) quality research, and all performance data contained in this publication was obtained in the specific operating environment and under the conditions described in this publication and is presented as an illustration only.

Performance obtained in other operating environments may vary and customers should conduct their own testing.

From within ISPF
Open the dataset: **<HLQ>.BNCHMRK5.JCL**
Again Edit the member: **SANDBOXI**
- From the command line type the following: **C SANDBOX SANDBOXI ALL** **SAVE** **SUB** -- and note the JobName produced by TSO.
From the command line, type: **=3.8**
Open the JES job name and Job number for the compile you just submitted (note that the JCL Job name is <HLQ...I>
M/PF8 to the bottom of the file
View the syntax return code
From the command line, type: **=1** - to bring up ISPF browse
Open the file: **<HLQ>.BNCHMRK5.LISTING(SANDBOXI)**
From the command line, type: M and press PF8
This will bring up all of the syntax errors in succession
With your cursor on the command line, press PF2
Open **<HLQ>.BNCHMRK5.COBOL(SANDBOXI)** in ISPF edit **(=2)**
Fix all of the syntax errors in **SANDBOXI** - by cross-referencing the compile listing with the **SANDBOXI** program. Because we don't want you to guess, here are the identifier name mis-spellings: VLAD-RECORD should be **VALID-RECORD** READ PATMST should be READ **PATMSTR** GO TO 100-ABEND-RTN should be **1000**-ABEND-RTN INPATIENT-REC-ERROR should be **INPATIENT-TREATMENT-REC-ERR**
When you've fixed all of the errors (or think you have, and are ready to re-compile), from the command line, type **SAVE** <Enter>
Press PF9 to swap to the split screen

Figure 12-1 Destail script for ISPF Tasks

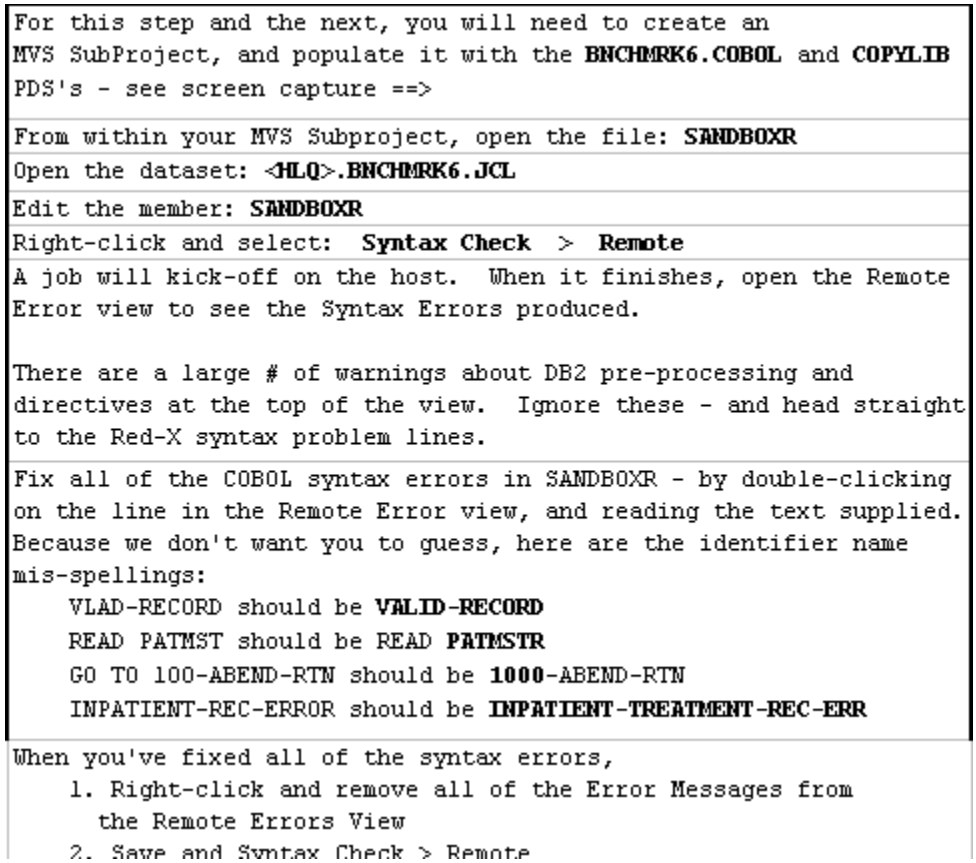


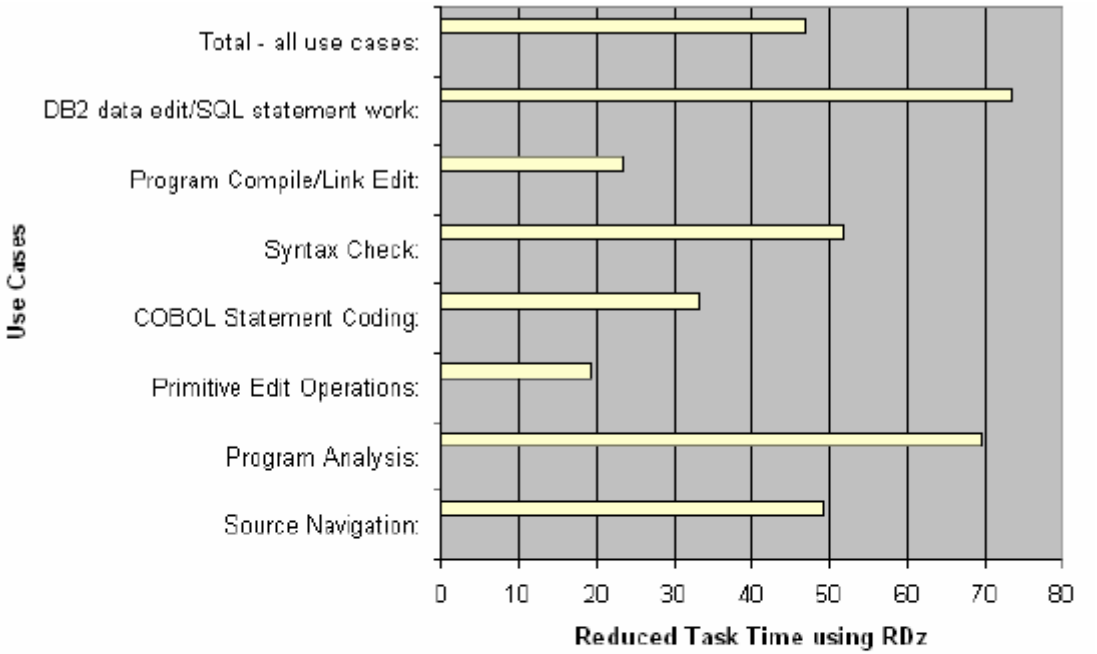
Figure 12-2 Detailed script - RDz Tasks

12.4 Task Summary Results – All Participant Subset

The data from individual timings (down to the second) for each participant doing each task were entered into a spreadsheet, and the results were then graphically summarized (see figure 12-3) as the percentage less time it took for all participants to complete the 100 tasks. In the bar chart, zero (0) represents the ISPF baseline default.

You may note that some categories of tasks showed more or less productivity, but overall, the results were favorable for RDz across the board.

We will analyze these results in a bit, but before doing that, it's worth noting that we also broke out the participants into a subset of what we called "TSO Top Gun" developers.



Use Case	% Less time to complete tasks with RDz
Source Navigation:	49.26
Program Analysis:	69.67
Primitive Edit Operations:	19.22
COBOL Statement Coding:	33.11
Syntax Check:	51.89
Program Compile/Link Edit:	23.38
DB2 data edit/SQL statement work:	73.41
Total - all use cases:	46.88

Figure 12-3 "All Participants" results - the percentage less time to complete the 100 tasks using RDz

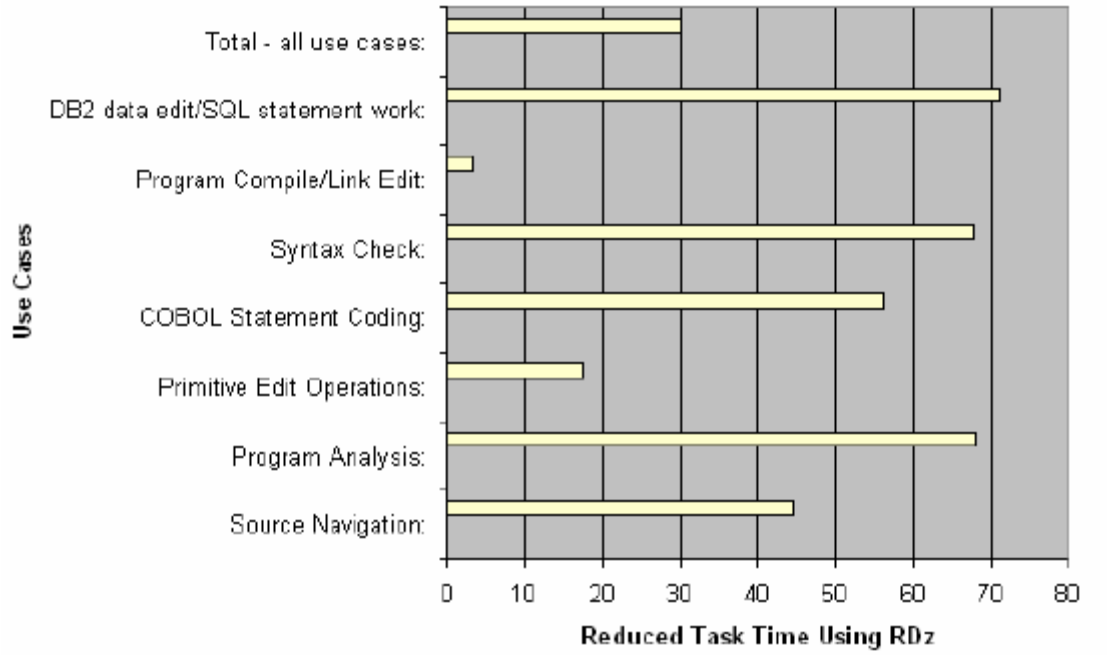
The TSO Top Gun developers were developers who characterized themselves as:

- Current with ISPF
- With at least 20 years of ISPF experience

- Had accurate and fast typing skills

We felt that these individuals represented a specific veteran ISPF-oriented constituent subset of most z/OS shops, so we created another set of summary statistics for them (see figure 12-4).

12.5 Task Summary - TSO “Top Gun” Participants



Use Case	% Less time to complete tasks with RDz
Source Navigation:	44.53
Program Analysis:	67.99
Primitive Edit Operations:	17.45
COBOL Statement Coding:	55.98
Syntax Check:	67.58
Program Compile/Link Edit:	3.37
DB2 data edit/SQL statement work:	70.92
Total - all use cases:	30.03

Figure 12-4 “TSO Top Gun” results - the percentage less time to complete the 100 tasks using RDz

From 12-4 you can see that the TSO “Top Gun” developers lived up to their nickhame. Their results were better (and often significantly better) in most work categories than the pool of all developers - which would represent in any given shop your Entry Level programmers combined with TSO Top Gun users.

While most shops sport a healthy mix of experienced and Entry Level programmers (and will more and more over the next 10 years as the Generation I programmers near retirement), the standard deviation for the Top Gun group was - across the board in the single digits and often as low as 2 or 3 (showing consistent and stastically significant results).

12.6 Analysis – and Feedback From Participants

After each person completed their work and returned the spreadsheet, we asked closed and open-ended questions regarding what they believed the reasons were for the RDz productivity. The results were a little surpring but uniform:

There were four sources of productivity:

1. (Significantly) less typing with RDz
2. RDz Advanced Tooling
3. Better use of Screen Real Estate
4. Responsive Desktop/Windows Environment

The reason these were “surprising” was that it’s usually assumed that the superior engineering that goes into new technology is the primary reason for improvements. At least in the case of the study the engineering was definitely a factor, but there were many other positive factors - or “unintended consequences” of Eclipse and RDz development.

12.6.1 (Significantly) less typing

Every activity on ISPF requires some degree of typing – and typically custom typing (unique Find/Change commands, line location, etc.). Even navigation is done with typing (=3.14, =P.DB2.3, etc.)

With RDz most of those same actions and developer activities have been encapsulated into Declarative Tooling (Context menus, Intelli-sense/language-sensitive editing, etc.) - and the typing differential (as emphasized by this section sub-header) is not a trival amount. Those participants that did both the ISPF and RDz scripts (and all 23 participants did both) consistently lised “less typing” as their number one reason for productivity.

As an example of this, with ISPF development activities that involve working with program copybooks/includes require significantly more effort than RDz - where there are context menu options used to open and browse copybooks.

12.6.2 Integrated/Language-Sensitive/Hyper-linked/Feature-rich Tooling

The green-screen (ISPF) development paradigm is a manual and linear development process model, and the tools require constant panel navigation in order to access needed functionality (more typing/more MIPS - through every PF-Key and Enter key pressed, etc.).

There is no hyper-linking of temporary results (for example, Search and Syntax errors), and no integration among or between the ISPF tools. DB2 Table editing is done by entering long and cumbersome SQL INSERT or UPDATE statements, etc.

RDz integrates almost all common development functionality into a single-system “concurrent development” view and the features and facilities needed to do something are available without navigation (context menus, etc.)

RDz hyper-links source results whenever possible, to further dial back superfluous navigation.

12.6.3 Use of Windows-graphical "Screen Real-Estate"

ISPF allows developers to see from 24 to 55 lines of source at a time – however there is a lot of wasted real estate and the fidelity of the source view is a problem when you work with 40 or higher lines..

RDz provides up to 190 lines of source viewing (and editing) in split-screen mode, with virtually no wasted real estate. This is because RDz allows vertical screen splitting - a more efficient way to handle 80-column COBOL statements.

With RDz, you can:

- Copy/Paste between open views of the same or different source files
- Edit the same program in two different areas simultaneously - which is useful and not possible for ISPF
- See the results of a change reflected across all open windows immediately

12.6.4 Responsive Desktop/Windows Environment

Even though the IBM mainframe provided sub-second response time for the tests (even the compile and link jobs finished in less than a second - for all participants), the ability to use the desktop environment (real-time Scroll bars, PgUp/PgDn etc.) was still appreciably - if not significantly faster for certain tasks standard programming tasks (like aligning code on-screen to specific statements, code navigation, etc.)

This was interesting to one of the authors as going back to the late 1980's when "downsizing" was the I/T mantra, it was assumed that mainframes were not "flexible and responsive - enough". At least in this study a germ of truth was found for this - although it was the smallest factor in the RDz benefits list.

12.7 Mitigating Factors

- The following must be noted about Benchmarks - in the spirit of transparent analysis:
- No use of custom ISPF Edit Macros
  - Many shops (and individual programmers within shops) have developed and use custom editing macros during their work
  - These macros would in all likelihood improve have improved the ISPF Benchmark results
  - To what degree is unknown…but possibly as much as: 5-10%
    - Note that the reason for not using any ISPF Edit Macros was that every shop's (and programmer's) macros are likely unique - and so the applicability of testing for a given unique macro would be low - to the vast majority of shops
    - However everyone uses ISPF options: 3.4, option 2, option 1, split-screen, TSO Job submission, etc.
- No use of custom RDz Macros:
  - In the same way, mature RDz shops have either re-created their Macro facilities in RDz, or have created their own unique extensions to the editor.
  - These would in all likelihood improve the RDz results as much as: 3 – 5%
    - As a short experiment - if you return to the results for the TSO Top Gun Developers (figure 12-4) and factor in a 10% improvement in ISPF productivity through macros then factor in a 3% improvement in RDz productivity (a 7% overall improvement) you will probably still find the results - a net: 23% improvement in productivity, conspicuous.

- Years of ISPF experience
  - The ISPF development experience (12.7 years) of the participants is considerably more than the RDz experience (1.3 years)
  - Howeve many shops have groups of developers with 20+ years of ISPF experience
  - You may wonder (as we did) how much more productive you get after almost 13 years of using a product - however, we felt this should be called to your attention.
  - This issue was also mitigated as far as possible through the use of the detailed:
    - ISPF script (down to the PF-Key to be pressed)
    - RDz script (down to the context-menu used)

12.8 Summary

In this section we have introduced you to a recent study done comparing two IBM products. We:

- Described the participant demograpics
- Detailed the methodology
- Presented the findings
- Analyzed the results

Again - if you're interested in obtaining the scripts used in the study please visit the ITSO site.