

Rewrite, Convert or Run Unchanged

How the migration R-words apply to batch, what the evidence says about large rewrites, and the three choices for every sort step

Bill Ahlbrandt

President and CEO, Ahlbrandt Software

Abstract

Every mainframe modernization program decides, job by job, whether to rewrite what it has, convert it or keep it. The “R” taxonomies from Gartner and AWS were written for applications, and they say very little about the batch utility steps that do a lot of the work overnight. In this paper I go through what each route means for JCL, for COBOL programs and for utility steps like sort, and look at the public evidence on large rewrites. Then I take the three options for a sort step in particular: rewrite it in the target language, convert its control statements to another tool’s syntax, or run the existing statements unchanged on an engine that accepts them. Everything a sort does is spelled out in a few control statements, so of course it can be rewritten. The question is whether that buys you anything that pays for the testing it creates.

What I found

- **The R taxonomies aren’t one standard.** Gartner’s five (2011), AWS’s six (2016, later seven) and Gartner’s later seven options use the same words for different things, “refactor” most of all.
- **Large rewrites have a poor track record, and it’s consistent.** In a vendor-commissioned 2024 survey, nine in ten rewrite projects didn’t succeed the first time. The US GAO found that only three of the ten critical federal legacy systems it flagged in 2019 had been modernized by 2025.
- **Utility steps are declarative.** A sort, copy or REPRO step does exactly what its control statements, data attributes and documented rules say. That makes it the cheapest part of a batch estate to move unchanged, and a surprisingly expensive part to rewrite.
- **You can test for the ways sort behavior drifts.** They’re specific: collation, packed-decimal signs, the order of equal keys, record formats, literal encoding, and installation defaults that aren’t in the JCL.
- **My rule of thumb:** fold a sort into new code only where the files on either side of it are being redesigned too. Where they survive, keep the control statements and change the engine.

1. Why the route matters

Not many mainframe shops are leaving altogether. In Kyndryl’s 2025 survey of 500 business and IT leaders, run by Coleman Parkes Research, exactly one respondent was trying to get off the mainframe entirely.

The average organization was moving 28 percent of its applications to other platforms, and 70 percent said they had trouble finding people with both mainframe and modern skills.¹ But 28 percent at a big bank or insurer can still mean thousands of batch jobs, and every one of them gets rewritten, converted or moved as it is. That choice, made job by job and step by step, decides most of the cost and most of the risk.

The pressure to choose isn't letting up. The US Government Accountability Office reported in July 2025 that the eleven federal legacy systems most in need of modernization (the oldest is about 60 years old) cost about \$754 million a year to operate and maintain, and that eight of them use outdated languages. Both Treasury systems in the sample run on COBOL and assembler.² Meanwhile AI-assisted conversion tools have made rewriting look cheaper than it ever has. Gartner warned in an April 2026 research note, quoted by a modernization vendor, that more than 70 percent of mainframe exit projects started in 2026 “will fail to produce the intended benefits due to an overestimation of generative AI tooling capabilities.”³

Sort sits right in the middle of this. Most batch streams sort, merge or copy data between programs, and under each route those utility steps behave differently from the application code around them.

2. What the R-words actually mean

People often quote the taxonomies as if they were one standard. There are three of them, and they disagree. In May 2011 Gartner described five ways to migrate applications to the cloud: **rehost**, **refactor**, **revise**, **rebuild** and **replace**.⁴ In Gartner's usage, *refactor* meant running an application on a cloud provider's platform, and *rebuild* meant throwing out the code and re-architecting it for that platform. In November 2016 Stephen Orban of AWS published six strategies that, he wrote, “build upon the 5 R's that Gartner outlined”: **rehost**, **replatform**, **repurchase**, **refactor/re-architect**, **retire** and **retain**.⁵ Here *refactor* means re-imagining the architecture, which is closer to Gartner's *rebuild*. AWS later added a seventh, *relocate*, and describes its mainframe service in two patterns: replatforming, which recompiles existing code, and automated refactoring, which transforms it into Java-based services.⁶ Exhibit 1 lines up the terms and shows what each route does to the three parts of a batch job.

Exhibit 1. What each route does to a batch job

Route (and aliases)	JCL	COBOL programs	Utility steps (sort, copy, REPRO)
Retain, or rehost to another IBM Z	Unchanged	Unchanged	Unchanged, on z/OS
Retire	Job removed	Removed	Removed, once nothing reads the output
Replatform (mainframe “rehost”, emulation)	Run by the platform's JCL support; minor edits	Recompiled with a compatible compiler	Supplied by the platform or a third-party engine; control statements can run unchanged
Automated refactor (conversion)	Translated into scripts in a target language	Translated to Java, C# or similar	Reimplemented in the converter's runtime, with its own coverage
Rewrite (rebuild, re-architect)	Redesigned as workflows or pipelines	Rewritten by hand or with AI help	Folded into SQL, dataframe or program logic, or no longer needed
Replace (repurchase)	The package's own scheduling	Replaced by a package	Survive in the interface and extract jobs around the package

Sources: Gartner (2011, and later seven-option framework); AWS (2016, 2021).^{4–7} Mapping to batch components by the author.

Gartner later published its own seven modernization options, ranked by how easy they are to implement: **encapsulate**, **rehost**, **replatform**, **refactor**, **rearchitect**, **rebuild** and **replace**. In that list, *refactor* restructures code “although not its external behavior”, while *rebuild* means to “redesign or rewrite the application component from scratch.”⁷ So one word has meant three different things. Mainframe usage adds a fourth. “Rehosting” COBOL and JCL usually means recompiling them for an emulation environment on x86, which AWS calls replatforming.

3. What the evidence says about large rewrites

There isn't much public evidence on mainframe rewrites, and a lot of what there is was paid for by vendors of the alternatives. Read with that in mind, it's consistent. A Forrester study commissioned by Rocket Software, which sells rehosting and modernization tools, surveyed 309 decision-makers in August 2024 and found that nine in ten mainframe application rewrite projects didn't succeed the first time. More than half the respondents said repeated failures had stalled wider transformation work.⁸ Standish Group data from its CHAOS 2020 research, published with Micro Focus, reported a 26 percent success rate and a 20 percent failure rate for "rip and replace" projects, against 71 percent and 1 percent for incremental modernization.⁹ Both sponsors sell the incremental route.

The official sources show the same pattern, with no commercial interest behind them. The Internal Revenue Service originally estimated that CADE, the system meant to replace its 1960s-era Individual Master File, would cost \$66 million and be ready by late 2001. After delays and overruns it stopped work in 2009. Its successor, CADE 2, started in 2010 with its first two phases due by January 2014. Full deployment is now projected for fiscal 2028.¹⁰ The IRS reported in 2022 that it had converted more than 90 percent of the core assembler code for individual returns to Java.¹¹ Translating the code turned out to be a lot faster than retiring the system it belongs to. Exhibit 2 collects the main data points.

Exhibit 2. The public record on large rewrites and replacements

Source (type)	What was examined	Finding
Forrester for Rocket Software, 2024 (vendor-commissioned survey, n = 309)	Mainframe application rewrite projects	Nine in ten didn't succeed the first time
Standish Group CHAOS 2020 data, with Micro Focus (vendor-sponsored)	"Rip and replace" versus incremental modernization	26% success, 20% failure, versus 71% success, 1% failure
Gartner, April 2026 (analyst prediction)	Mainframe exit projects started in 2026	More than 70% will fail to produce intended benefits
US GAO, GAO-25-107795, July 2025 (official)	Ten critical legacy systems flagged in GAO-19-471 (2019)	Three modernized by February 2025; three of the eleven now flagged have complete plans
Congressional Research Service, IF12525, 2024 (official)	IRS CADE and CADE 2	\$66m estimate, due 2001, stopped 2009; successor now due fiscal 2028
UK FCA and PRA, December 2022 (official)	TSB migration to a new IT platform, April 2018	Disruption until December 2018; £48.65m in fines; £32.7m redress

Sources: references 2, 3, 8–10, 12. Vendor-sponsored figures are indicative, and definitions of success vary.

TSB was a platform migration. There was no code rewrite. The regulators' finding applies to both anyway: the bank "failed to organise and control the IT migration programme adequately."¹² None of these failures turned on sort. They turned on how much was changed at once, and how much of the old behavior was never written down. That's exactly where utility steps are different from application code.

4. Why utility steps convert differently

A COBOL program's behavior lives in its procedure division, in its copybooks, in the compiler options that govern truncation and rounding, and in decades of changes nobody wrote down the reasons for. Recovering that behavior is the expensive part of any rewrite. A utility step is a different animal. What a sort step does is completely determined by three things: its **control statements** (SORT, INCLUDE, OUTREC and so on), the **attributes of its data** (record format, record length, character encoding and field types), and the utility's **documented rules**, which IBM publishes in the DFSORT Application Programming Guide.¹³ The step is declarative. It says what result you want and leaves the how to the sort.

There's one catch that matters in practice. Some of those rules are set by installation defaults and never show up in the job. DFSORT's installation options, for example, decide whether records with equal keys keep their input order (EQUALS) unless a job says otherwise, and IBM ships the product with that option off.¹³ An inventory that only reads the JCL and control statements will miss them.

There are also a lot of utility steps. Besides the explicit sort and merge steps, many shops route IEBGENER copies through the sort product (DFSORT's ICEGENER facility is there for exactly that), and IDCAMS REPRO moves data between sequential and VSAM files.¹³ We couldn't find an independent published figure for the share of job steps that run utilities. Conversion vendors just list sort, IEBGENER and IDCAMS among the utilities every conversion has to handle.¹⁴ In the migration inventories I've seen, utility steps often outnumber application program steps. Any shop can get its own number quickly by counting program names (SORT, ICEMAN, SYNC SORT, IEBGENER, ICEGENER, IDCAMS and the rest) in its JCL or in SMF type 30 step records.

5. Three options for a sort step

Whatever route the application around it takes, every sort step ends up one of three ways.

- **Rewritten.** The sort, selection and reformatting are written in the target language: an ORDER BY in SQL, a dataframe operation, a few lines of Java or Python. This is where a rewrite naturally ends up, and it's what AI-assisted conversion usually produces.
- **Converted.** The control statements are translated, by a tool or by hand, into another sort product's syntax or a converter's runtime library. One conversion vendor says its sort component delivers "about 90 percent of the functions" of DFSORT or Syncsort,¹⁴ which is an honest way of raising the question every conversion raises: which ten percent does your shop use?
- **Run unchanged.** The existing statements are run by a sort engine on the new platform that accepts mainframe syntax. Emulation environments ship one. Rocket Enterprise Server (formerly Micro Focus) includes a DFSORT-compatible utility called MFSORT,¹⁵ and there are third-party engines for Linux, Unix and Windows.

Exhibit 3 takes one ordinary DFSORT step through each option. The step reads fixed-length 80-byte EBCDIC records, keeps the ones whose region code is EU, sorts them by account number ascending and by a six-byte packed-decimal amount descending, and writes out just those two fields.

Where the behavior drifts

That little example has most of the ways a rewritten or converted sort goes quietly wrong. You won't see any of them in a code review. A byte-for-byte comparison shows every one.

- **Collation.** In EBCDIC, letters sort before digits. In ASCII and most database collations, digits come first. An account "A100" comes before "1000" on the mainframe and after it in a naive port. The Python in Exhibit 3 only gets this right because it compares raw bytes.
- **Packed-decimal signs.** DFSORT treats sign nibbles C, A, E and F as positive and D and B as negative.¹³ Hand-written decoders often only recognize C and D, and they fail on the rest or put them in the wrong order.
- **Equal keys.** Unless EQUALS is on, the order of records with identical keys isn't guaranteed. A rewrite that keeps input order produces a different file that's just as valid, and then every regression comparison of that file fails for a reason that isn't a defect.
- **Literals and encodings.** C'EU' in a control statement is compared with EBCDIC data. On an ASCII platform, the statement still has to mean EBCDIC "EU" if the data was moved in binary, which packed and binary fields require.
- **Record formats and edge cases.** Variable-length records with four-byte descriptors, short records, empty inputs, arithmetic overflow in SUM and century windowing in date fields all have documented DFSORT behavior, and a rewrite has to reproduce each one on purpose.

Exhibit 3. One DFSORT step, three ways (worked example)

Approach	What the step becomes	What the team now owns
Original, run unchanged	<pre>INCLUDE COND=(21,2,CH,EQ,C'EU') SORT FIELDS=(1,10,CH,A,31,6,PD,D) OUTREC BUILD=(1,10,31,6)</pre>	The same three statements. The engine has to read EBCDIC data, treat the literal as EBCDIC and follow DFSORT's rules and the site's defaults.
Rewritten in Python	<pre>def pd(b): # packed decimal -> int h = b.hex(); v = int(h[:-1]) return -v if h[-1] in 'bd' else v src = open('txn.dat', 'rb').read() recs = [src[i:i+80] for i in range(0, len(src), 80)] recs = [r for r in recs if r[20:22] == b'\xc5\xe4'] recs.sort(key=lambda r: (r[0:10], -pd(r[30:36]))) out = b''.join(r[0:10] + r[30:36] for r in recs)</pre>	Custom code for decoding, collation and record handling. It's only correct because it compares raw EBCDIC bytes, with "EU" hard-coded as EBCDIC X'C5E4'. Python's sort is stable. DFSORT's default isn't.
Rewritten in SQL	<pre>SELECT acct, amount FROM txn WHERE region = 'EU' ORDER BY acct ASC, amount DESC;</pre>	Needs a load step to decode EBCDIC and packed fields first, and an unload step to rebuild the records. ORDER BY follows the database collation, which isn't EBCDIC.
Converted (generic target syntax, illustrative)	<pre>input: txn.dat fixed 80 ebcdic fields: acct 1:10 char, region 21:2 char, amount 31:6 packed filter: region == "EU" keys: acct asc, amount desc output: txn_eu.dat [acct, amount]</pre>	A second dialect to learn and test. Only correct if the target tool supports every type, option and default used.

Record layout assumed: account number in bytes 1–10, region code in 21–22, amount in 31–36 as packed decimal. The Python was tested against EBCDIC sample data. The conversion syntax is made up and doesn't represent any particular product.

6. The trade-offs

None of the three is right everywhere. Exhibit 4 compares them. Two things stand out. First, the cost of a rewritten sort is in the testing. Writing the Python in Exhibit 3 takes minutes. Proving it matches the original across every data condition takes far longer, and you have to do that for each of maybe thousands of steps. Second, keeping the control statements makes parallel running cheap. The same statements can run on z/OS and on the new platform against the same input, and you compare the outputs byte for byte with nothing in between. That's what lets a migration fall back to the mainframe on a bad night without reconciling two versions of the logic.

Exhibit 4. Rewrite, convert or run unchanged, side by side

	Rewrite	Convert	Run unchanged
Testing burden	Highest: every step is new code, and outputs often change format too	High: every statement translated; coverage gaps turn up in testing	Lowest: prove the engine once, then compare outputs byte for byte
Skills	Target-language developers who also understand EBCDIC, packed decimal and DFSORT rules	Both dialects, during and after conversion	DFSORT syntax, widely known and documented by IBM
Lock-in	None to a sort product; to custom code instead	To the target tool and its dialect	To a dialect with several implementations; check each engine's coverage
Behavior drift	Most exposed (collation, signs, ties)	Exposed where tool semantics differ	Limited to the engine's documented differences
Mainframe fallback and parallel run	Two code bases to keep in step	Two dialects to keep in step	Same statements run on both platforms
Best fit	Data redesigned around the step (tables, streams)	A shop standardizing on a tool it already runs	Files and neighboring programs survive the move

Assessment by the author.

Rewriting makes real sense where the data design is changing. If the files before and after a sort are being replaced by database tables or event streams, the sort has no file to read and nothing to feed, and it should dissolve into the new design. Rewriting it faithfully would keep a step the new architecture doesn't need. Converting makes the most sense where an organization already runs another sort product as a standard on its target platform, and has checked that product's coverage against the statements its jobs actually use.

7. How to decide, step by step

- 1 Inventory the steps.** Count steps by program name from the JCL and SMF type 30 records, and list every distinct control statement, option and data type in use. Include the installation's sort options.
- 2 Retire first.** Trace which outputs are still read. A step whose output nobody reads doesn't need a decision at all.
- 3 Follow the data.** For each sort that's left, ask whether the files on both sides survive the move. If they do, the sort is a candidate to run unchanged, whatever happens to the programs around it.
- 4 Rewrite only with the data design.** Fold a sort into new code where its input and output are being redesigned, and budget for equivalence testing on the downstream results.
- 5 Convert only against a checked list.** Check each statement and option in the inventory against the target tool's documented coverage before you commit. Finding the gaps in testing is too late.
- 6 Prove it on hard data.** Compare outputs byte for byte on test files built to include mixed letters and digits in keys, negative and oddly signed packed fields, duplicate keys, short and variable-length records, and empty inputs.

8. Bottom line

- **Define the words.** Rehost, refactor and rebuild have meant different things in the frameworks that made them popular. If a program doesn't pin them down, it can price one thing and deliver another.
- **The evidence favors smaller changes.** Vendor-sponsored and official sources both show large rewrites failing or running far past plan. The failures turn on how much undocumented behavior gets changed at once.
- **Sort is different.** Everything it does is written down, so it's easy to rewrite and just as easy to keep. Nearly all the cost of rewriting it is the cost of proving the rewrite is right.
- **Let the data decide.** Where the files survive the move, I'd keep the control statements and change the engine. Where the data is redesigned, let the sort dissolve into the new design and test the results.

About the author

Bill Ahlbrandt is President and CEO of Ahlbrandt Software, which he founded in 1993. Before that he was a systems programmer, Director of Information Security and Director of R&D at Acxiom. He has been working on mainframe sort and batch for more than thirty years, and his company's sort engine runs mainframe sort jobs on Linux and other platforms for banks, insurers and government agencies.

Full disclosure: my company sells sort software for platforms other than the mainframe. It runs mainframe sort control statements as they are, which is one of the three options in this paper.

References

1. Kyndryl, *State of Mainframe Modernization Survey*, 9 September 2025 (n = 500; fieldwork by Coleman Parkes Research). Vendor-sponsored.
2. US Government Accountability Office, *Information Technology: Agencies Need to Plan for Modernizing Critical Decades-Old Legacy Systems*, GAO-25-107795, July 2025.
3. Gartner, *Too Big to Fail: Why Mainframe Exit Projects Are Likely to Fail in the Age of Generative AI*, April 2026, as quoted by Rocket Software, “Mainframe modernization: the risks of AI rewrites”, undated.
4. Gartner, “Gartner Identifies Five Ways to Migrate Applications to the Cloud” (R. Watson), press release, May 2011; reported by Tech Monitor, 17 May 2011.
5. S. Orban, “6 Strategies for Migrating Applications to the Cloud”, AWS Enterprise Collection, 1 November 2016.
6. J. Allen, “7 Strategies for Migrating Applications to the Cloud, introducing AWS Mainframe Modernization and AWS Migration Hub Refactor Spaces”, AWS Executive in Residence blog, 30 November 2021.
7. Gartner, “7 Options to Modernize Legacy Systems”, Smarter With Gartner (quoting S. van der Zijden), undated online article.
8. Forrester Consulting, study commissioned by Rocket Software, August 2024 (n = 309); reported by CIO Dive, “Most mainframe application rewrites fail the first time”, 24 September 2024. Vendor-commissioned.
9. Micro Focus, “New multi-decade research indicates continual modernization programs are key to IT success”, press release on Standish Group CHAOS 2020 data, 2 December 2020. Vendor-sponsored.
10. G. Guenther, *The IRS Technology Modernization Program: An Overview*, Congressional Research Service IF12525, updated 20 May 2024.
11. Internal Revenue Service, “Modernizing tax processing systems”, 28 July 2022.
12. Financial Conduct Authority, “TSB fined £48.65m for operational resilience failings”, press release, 20 December 2022; joint action with the Prudential Regulation Authority.
13. IBM, *z/OS DFSORT Application Programming Guide*, SC23-6878: SORT, INCLUDE, OUTREC and OPTION statements, packed-decimal sign handling, EQUALS, ICEGENER; and *z/OS DFSORT Installation and Customization*.
14. SoftwareMining, “JCL to Unix shell script and Windows batch file translation”, service page, undated. Vendor source.
15. Rocket Software (formerly Micro Focus), Enterprise Server documentation: MFSORT utility.