Streamlining Statistical Programming with R: ADaM and TLF Validation Through a Compliant R Programming Environment

Mangesh Kalsekar, Bristol Myers Squibb, Hyderabad, India

ABSTRACT

The growing use of R in Statistical Programming reflects the industry's shift toward flexible, open-source tools for data analysis and reporting. While R offers clear advantages for ADaM dataset and TLF validation, its adoption in GxP-regulated environments requires careful planning, governance, and validation. This paper outlines a practical framework for building a robust, compliant R programming environment designed to support the day-to-day work of statistical programmers in regulated settings.

We use **Posit Workbench** for secure access, **renv** for version-locked package control, and **logr** for audit-ready logging. Modular functions and standardized templates simplify dataset transformation, QC, and TFL's. Tools like **arsenal** and **summarytools** support fast data comparisons and automated review outputs.

We aligned with Open-Source Governance and IT to enable risk-based package validation, infrastructure support, and compliance with regulatory expectations. This approach illustrates effective scalability of R within Statistical Programming teams, driving automation, enhancing transparency, and operational excellence in regulatory submissions.

INTRODUCTION

Statistical programming organizations are expected to improve efficiency, transparency, and reproducibility while maintaining compliance with regulatory requirements. R has emerged as a powerful alternative to traditional statistical programming tools due to its flexibility, extensibility, and robust open-source ecosystem. However, without appropriate governance, the use of R can introduce variability, limit auditability, and raise compliance and regulatory concerns.

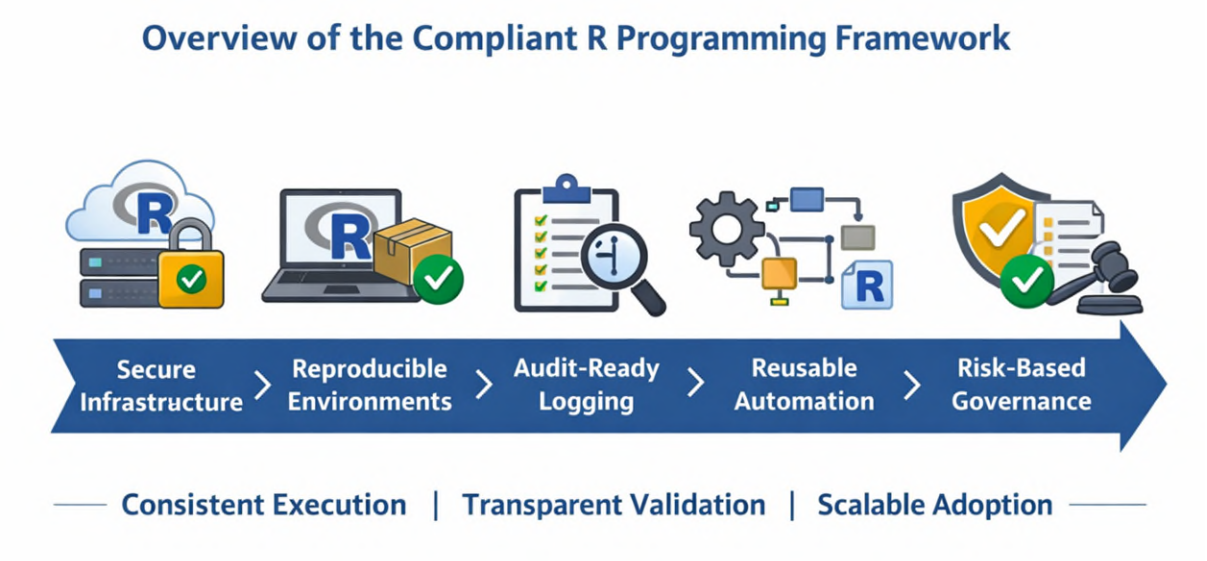
To enable broader adoption, R must be embedded within a controlled ecosystem that supports reproducibility, traceability, and standardized validation practices. This paper presents a practical framework that enables statistical programmers to use R confidently for ADaM and TLF validation in regulated environments.

CHALLENGES OF USING R IN REGULATED SETTINGS

Key barriers to the use of R in regulated statistical programming environments include inconsistent package versioning, lack of standardized logging and traceability, limited native validation capabilities, and uncertainty surrounding open-source governance. In the absence of defined controls, reproducibility of historical outputs and readiness for regulatory inspection are compromised. Consequently, R has traditionally been limited to exploratory analyses and non-submission activities

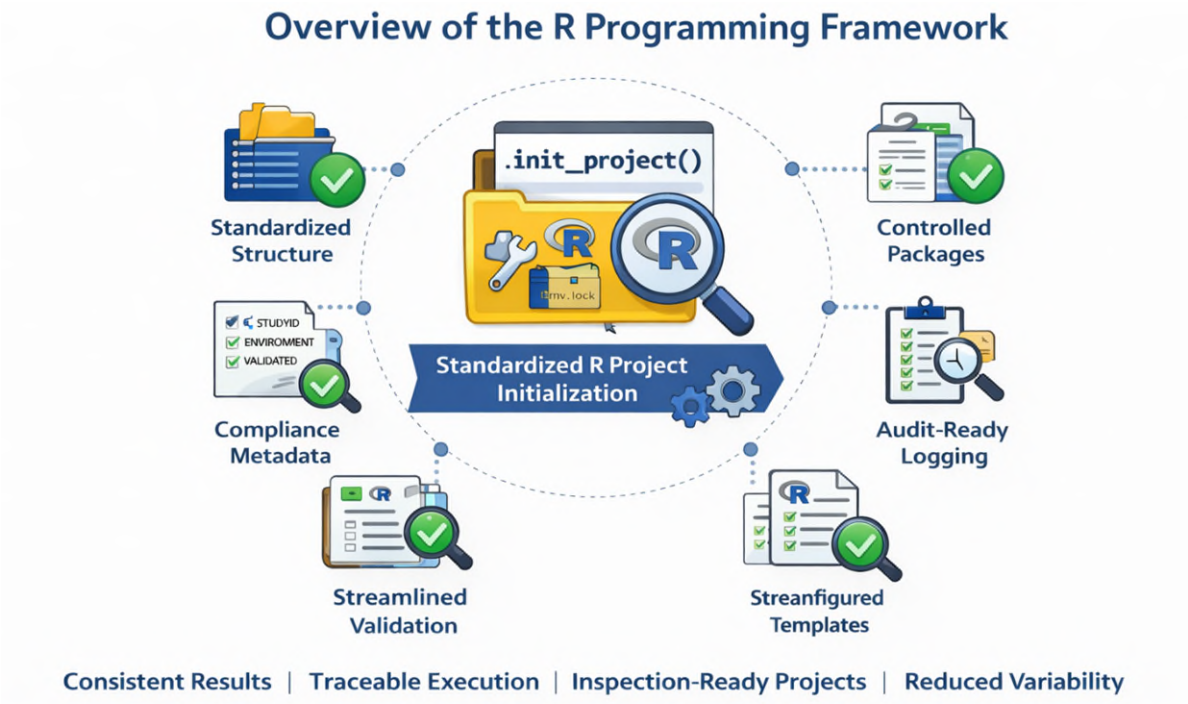
OVERVIEW OF THE COMPLIANT R PROGRAMMING FRAMEWORK

The framework creates a controlled R ecosystem with secure infrastructure, reproducible environments, audit-ready logging, reusable automation, and risk-based governance. Posit Workbench provides a centralized, secure development platform with consistent R versions and enterprise controls. Reproducibility is ensured through renv, which locks package versions for each study. Regulatory needs are met through structured logging with logr, capturing execution details and creating a transparent audit trail similar to SAS logs.



OVERVIEW OF THE R PROGRAMMING FRAMEWORK

The figure shows a regulated R framework built on controlled project setup, reproducible packages, audit ready logging, reusable automation, and compliance metadata, enabling consistent, inspection ready workflows across teams



PRACTICAL TECHNICAL EXAMPLE: ADaM VALIDATION WORKFLOW USING THE R PROGRAMMING WORKFLOW

Scenario

A programmer needs to validate ADSL and ADAE produced in a development environment against a validated ("reference") environment. The goal is to ensure

- reproducibility of the R environment,
- a complete audit trail, and
- standardized comparison outputs that can be reviewed and archived.

1) Secure Infrastructure: Controlled Execution Context

What it means technically: the code is executed in a centrally managed environment (e.g., Posit Workbench) with controlled R versions and OS-level access. From a workflow perspective, you treat the environment as "trusted," and you make your project self-contained (no user-specific hard-coded paths).

Practical control you implement in code: enforce root paths and fail fast if paths are not available.

```
stop_if_missing_path <- function(path) {
  if (!dir.exists(path)) stop("Required path not found: ", path, call. = FALSE)
}
```

2) Reproducible Environments: *renv* Lock + Restore

Goal: anyone rerunning the validation later (or during an audit) uses the exact same package versions.

Typical pattern:

- Create the project with renv
- Restore package versions before executing validation
- Snapshot only after approved upgrades

```
# One-time setup (project creation)
renv::init()
# Every run (ensures reproducibility)
renv::restore(prompt = FALSE)
# Controlled upgrade cycle (only when approved)
# renv::snapshot()
```

3) Audit-Ready Logging: Standardized Log File per Run

Goal: Every run creates a traceable log with inputs, outputs, timing, and warnings.

A practical logging pattern is to log:

- Project metadata (study id, run id, user, date/time)
- Input dataset locations
- Dataset list processed
- Summary of compare results

```
library(logr)
start_audit_log <- function(study_id, run_id = format(Sys.time(), "%Y%m%d_%H%M%S")) {
  dir.create("logs", showWarnings = FALSE, recursive = TRUE)
  logr::log_open(path = "logs", name = paste0("validation_", study_id, "_", run_id), level
= "INFO")
}
lg <- start_audit_log("PILOT001")
logr::log_print(lg, "Validation run started")
logr::log_print(lg, paste("R version:", R.version.string))
logr::log_print(lg, "renv restored successfully")
```

4) Standardized Structure: Predictable Locations for Inputs and Outputs

Goal: avoid ad hoc folders. Make validation “plug-and-play.”

Example directory convention :

- data/dev/adam/ (development datasets)
- data/val/adam/ (validated datasets)
- outputs/compare/ (comparison reports)
- outputs/metadata/ (metadata exports)
- logs/ (audit logs)

```
paths <- list(
  dev_adam = "data/dev/adam",
  val_adam = "data/val/adam",
  out_cmp  = "outputs/compare"
)
lapply(paths, function(p) if (!dir.exists(p)) dir.create(p, recursive = TRUE, showWarnings
= FALSE))
```

5) Reusable Automation: Dataset Discovery Without Hardcoding Extensions

Goal: user says “compare ADSL” and the framework finds the file regardless of .xpt / .sas7bdat / .csv (depending on your controlled formats).

```
find_dataset_file <- function(dir_path, domain) {
  candidates <- list.files(dir_path, pattern = paste0("^", domain, "\\."), full.names =
TRUE, ignore.case = TRUE)
  if (length(candidates) == 0) stop("No matching file found for: ", domain, " in ",
dir_path, call. = FALSE)
  if (length(candidates) > 1) stop("Multiple matches found for: ", domain, " in ",
dir_path, call. = FALSE)
  candidates[[1]]
}
```

6) Compare Report: PROC COMPARE–Like Output with Review-Ready Summary

Goal: produce a consistent report that reviewers can interpret quickly:

- metadata differences (variables missing/extra/type mismatches)
- record differences by keys
- value differences (counts + examples)

A practical implementation uses ***arsenal::comparedf*** for comparison plus a standardized output wrapper.

```
library(arsenal)
library(dplyr)

compare_report <- function(dev_df, val_df, keys) {
  cmp <- comparedf(dev_df, val_df, by = keys)
  s <- summary(cmp)
  list(
    comparedf_object = cmp,
    summary = s,
    n_obs = s$observations,
    n_values = s$values,
    attributes = s$attributes
  )
}
```

Example: Compare ADSL (DEV vs VAL)

```
library(haven)

read_dataset <- function(path) {
  ext <- tolower(tools::file_ext(path))
  if (ext == "xpt") return(haven::read_xpt(path))
  if (ext == "sas7bdat") return(haven::read_sas(path))
  if (ext == "csv") return(read.csv(path))
  stop("Unsupported extension: ", ext, call. = FALSE)
}

domain <- "adsl"
dev_file <- find_dataset_file(paths$dev_adam, domain)
val_file <- find_dataset_file(paths$val_adam, domain)

logr::log_print(lg, paste("DEV file:", dev_file))
logr::log_print(lg, paste("VAL file:", val_file))

adsl_dev <- read_dataset(dev_file)
adsl_val <- read_dataset(val_file)

res <- compare_report(adsl_dev, adsl_val, keys = c("STUDYID", "USUBJID"))

logr::log_print(lg, paste("Attribute diffs:", res$attributes))
logr::log_print(lg, paste("Observation diffs:", res$n_obs))
logr::log_print(lg, paste("Value diffs:", res$n_values))
```

7) Standard Output Artifact: Save Summary + Exceptions

Goal: generate a single summary file plus optional “top differences” listing for reviewers.

```
write_compare_summary <- function(domain, res, out_dir = "outputs/compare") {
  dir.create(out_dir, showWarnings = FALSE, recursive = TRUE)
  summary_df <- data.frame(
    domain = domain,
    attribute_differences = res$attributes,
    observation_differences = res$n_obs,
    value_differences = res$n_values
  )
  out_file <- file.path(out_dir, paste0(domain, "_compare_summary.csv"))
  write.csv(summary_df, out_file, row.names = FALSE)
  out_file
}
out <- write_compare_summary("adsl", res)
logr::log_print(lg, paste("Compare summary written:", out))
```

Finally close the log:

```
logr::log_print(lg, "Validation run completed")
logr::log_close(lg)
```

What this example demonstrates (mapped to your framework icons)

- **Secure infrastructure:** controlled runtime + path validation
- **Reproducible environments**: *renv::restore()* ensures exact versions
- **Audit-ready logging:** *logr* captures run metadata and results
- **Reusable automation:** dataset discovery + standardized I/O patterns
- **Risk-based governance:** framework components validated once, reused per study

Below is a one-command wrapper function we can create to Streamline the validation.

It runs the full workflow end-to-end and generates:

- Per-domain compare summaries (CSV)
- Combined dashboard CSV (one file across all domains)
- Single audit log for the entire run
- Optional “examples of diffs” file per domain (top mismatches) if you choose to enable it

One-command function: `run_adam_validation()`

```
run_adam_validation <- function(
  study_id,
  domains,
  keys_map,
  dev_adam_dir = "data/dev/adam",
  val_adam_dir = "data/val/adam",
  out_dir      = "outputs/compare",
  log_dir      = "logs",
  run_id       = format(Sys.time(), "%Y%m%d_%H%M%S"),
  write_examples = TRUE,
  max_examples   = 25
) {
  # ---- dependencies ----
  if (!requireNamespace("arsenal", quietly = TRUE)) {
    stop("Package 'arsenal' is required.", call. = FALSE)
  }
  if (!requireNamespace("logr", quietly = TRUE)) {
    stop("Package 'logr' is required.", call. = FALSE)
  }
  if (!requireNamespace("dplyr", quietly = TRUE)) {
    stop("Package 'dplyr' is required.", call. = FALSE)
  }
  if (!requireNamespace("tools", quietly = TRUE)) {
    stop("Package 'tools' is required.", call. = FALSE)
  }
# ---- helpers ----
  stop_if_missing_dir <- function(path) {
    if (!dir.exists(path)) stop("Required directory not found: ", path, call. = FALSE)
  }

  ensure_dir <- function(path) {
    if (!dir.exists(path)) dir.create(path, recursive = TRUE, showWarnings = FALSE)
    invisible(path)
  }
```

```
find_dataset_file <- function(dir_path, domain) {
    # Looks for domain.ext with any ext; picks first match
    hits <- list.files(
      dir_path,
      pattern = paste0("^", domain, "\\.(xpt|sas7bdat|csv|rds)$"),
      ignore.case = TRUE,
      full.names = TRUE
    )
    if (length(hits) == 0) {
      stop("No dataset file found for domain '", domain, "' in ", dir_path, call. = FALSE)
    }
    if (length(hits) > 1) {
      stop("Multiple dataset files found for domain '", domain, "' in ", dir_path,
           ". Keep only one (e.g., adsl.xpt).", call. = FALSE)
    }
    hits[[1]]
  }

read_dataset <- function(path) {
    ext <- tolower(tools::file_ext(path))
    if (ext == "xpt") {
      if (!requireNamespace("haven", quietly = TRUE)) stop("Package 'haven' is required for
XPT.", call. = FALSE)
      return(haven::read_xpt(path))
    }
    if (ext == "sas7bdat") {
      if (!requireNamespace("haven", quietly = TRUE)) stop("Package 'haven' is required for
SAS7BDAT.", call. = FALSE)
      return(haven::read_sas(path))
    }
    if (ext == "csv") {
      return(utils::read.csv(path, stringsAsFactors = FALSE, check.names = FALSE))
    }
    if (ext == "rds") {
      return(readRDS(path))
    }
    stop("Unsupported file extension: ", ext, call. = FALSE)
  }

safe_summary_extract <- function(cmp_obj) {
    s <- summary(cmp_obj)
    # arsenal summary structure can vary slightly; use defensive extraction
    get_num <- function(x) if (is.null(x)) NA_integer_ else as.integer(x)
    list(
      attributes   = get_num(s$attributes),
      observations = get_num(s$observations),
      values       = get_num(s$values)
    )
  }
write_csv <- function(df, path) {
    utils::write.csv(df, path, row.names = FALSE)
    invisible(path)
  }
  # ---- validate inputs ----
  stop_if_missing_dir(dev_adam_dir)
  stop_if_missing_dir(val_adam_dir)
  ensure_dir(out_dir)
  ensure_dir(log_dir)

  # Ensure keys_map covers requested domains
  missing_keys <- setdiff(domains, names(keys_map))
  if (length(missing_keys) > 0) {
    stop("keys_map missing entries for: ", paste(missing_keys, collapse = ", "), call. =
FALSE)
  }
```

```
# ---- open audit log ----
  log_name <- paste0("adam_validation_", study_id, "_", run_id)
  lg <- logr::log_open(path = log_dir, name = log_name, level = "INFO")
  on.exit(try(logr::log_close(lg), silent = TRUE), add = TRUE)

  logr::log_print(lg, paste0("Validation run started | study_id=", study_id, " | run_id=",
run_id))
  logr::log_print(lg, paste("R:", R.version.string))
  if (requireNamespace("renv", quietly = TRUE)) {
    # Not mandatory, but helpful for audits
    lock_exists <- file.exists("renv.lock")
    logr::log_print(lg, paste("renv.lock present:", lock_exists))
  } else {
    logr::log_warn(lg, "Package 'renv' not found; reproducibility evidence limited to
session info.")
  }

  dashboard <- list()
  # ---- main loop ----
  for (domain in domains) {
    keys <- keys_map[[domain]]
    logr::log_print(lg, paste0("---- Domain: ", domain, " | Keys: ", paste(keys, collapse =
", "), " ----"))
    dev_file <- find_dataset_file(dev_adam_dir, domain)
    val_file <- find_dataset_file(val_adam_dir, domain)

    logr::log_print(lg, paste("DEV file:", dev_file))
    logr::log_print(lg, paste("VAL file:", val_file))

    dev_df <- read_dataset(dev_file)
    val_df <- read_dataset(val_file)

    # Ensure keys exist
    missing_in_dev <- setdiff(keys, names(dev_df))
    missing_in_val <- setdiff(keys, names(val_df))
    if (length(missing_in_dev) > 0 || length(missing_in_val) > 0) {
      msg <- paste0(
        "Key variables missing. DEV missing: ",
        paste(missing_in_dev, collapse = ", "),
        " | VAL missing: ",
        paste(missing_in_val, collapse = ", ")
      )
      logr::log_error(lg, msg)
      stop(msg, call. = FALSE)
    }
```

```
cmp <- arsenal::comparedf(dev_df, val_df, by = keys)
    counts <- safe_summary_extract(cmp)
    # Per-domain summary
    domain_summary <- data.frame(
      study_id               = study_id,
      run_id                 = run_id,
      domain                 = domain,
      dev_file               = dev_file,
      val_file               = val_file,
      key_vars               = paste(keys, collapse = ", "),
      attribute_differences  = counts$attributes,
      observation_differences= counts$observations,
      value_differences      = counts$values,
      stringsAsFactors       = FALSE
    )
    # Write per-domain summary CSV
    per_domain_path <- file.path(out_dir, paste0(domain, "_compare_summary_", run_id,
".csv"))
    write_csv(domain_summary, per_domain_path)
    logr::log_print(lg, paste("Per-domain summary written:", per_domain_path))

    # Optional: write examples of diffs (top rows)
    if (isTRUE(write_examples)) {
      # Create a lightweight "examples" output
      ex_path <- file.path(out_dir, paste0(domain, "_diff_examples_", run_id, ".txt"))
      sink(ex_path)
      cat("Compare diff examples\n")
      cat("Study:", study_id, "\nDomain:", domain, "\nRun:", run_id, "\n\n")
      print(cmp, max.rows = max_examples)
      sink()
      logr::log_print(lg, paste("Diff examples written:", ex_path))
    }
    dashboard[[domain]] <- domain_summary
    logr::log_print(lg, paste0("Domain complete: ", domain,
                               " | attr=", counts$attributes,
                               " | obs=", counts$observations,
                               " | values=", counts$values))
  }
  # ---- combined dashboard ----
  dashboard_df <- do.call(rbind, dashboard)
  dashboard_path <- file.path(out_dir, paste0("adam_validation_dashboard_", study_id, "_",
run_id, ".csv"))
  write_csv(dashboard_df, dashboard_path)
  logr::log_print(lg, paste("Dashboard written:", dashboard_path))

  logr::log_print(lg, "Validation run completed successfully")

  invisible(list(
    study_id = study_id,
    run_id = run_id,
    dashboard_path = dashboard_path,
    per_domain = dashboard_df,
    log_file = file.path(log_dir, paste0(log_name, ".log"))
  ))
}
```

Example usage (your one-command call)

```
run_adam_validation(
  study_id = "CDSIC001",
  domains  = c("adsl", "adae"),
  keys_map = list(
    adsl = c("STUDYID", "USUBJID"),
    adae = c("STUDYID", "USUBJID", "AESEQ")
  )
)
```

What it generates (typical outputs)

In outputs/compare/:

- adsl_compare_summary_<runid>.csv
- adae_compare_summary_<runid>.csv
- adam_validation_dashboard_CDSIC001_<runid>.csv
- (optional) adsl_diff_examples_<runid>.txt, adae_diff_examples_<runid>.txt

In logs/:

- adam_validation_CDSIC001_<runid>.log

"We standardized ADaM & TFL validation to a single command that restores a controlled R environment, runs dataset comparisons with defined keys, writes review-ready summaries, and produces a single audit log for traceability."

CONCLUSION

This paper presented a practical approach for enabling the use of R in regulated statistical programming through a controlled, reproducible, and auditable framework. By combining secure infrastructure, version-locked environments, standardized logging, and reusable automation, the framework addresses key compliance concerns that have historically limited the use of R in regulatory workflows.

The ADaM validation use case demonstrated how standardized R workflows can deliver reproducible results, transparent audit trails, and review-ready comparison outputs in a single controlled run. With appropriate governance and validation applied at the framework level, R can be scaled beyond exploratory use to support routine ADaM and TLF validation activities within statistical programming organizations.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Mangesh Kalsekar
Bristol Myers Squibb
Hyderabad, India
Email: mangesh.kalsekar@bms.com