Реинжиниринг реконструкции ливней в эксперименте Baikal-GVD: принципы работы с унаследованным научным ПО

А.Г. Соловьев, Ж.-А.М. Джилкибаев, В.Я. Дик, Т.В. Елжов, Б.А. Шайбонов

***Аннотация* — В статье описывается реинжиниринг унаследованного фортранного кода реконструкции ливней заряженных частиц для эксперимента Baikal-GVD и его интеграция в современный фреймворк BARS, написанный на C++. Вместо автоматической конвертации применен ручной поэтапный перевод с верификацией на каждом шаге. Общие блоки (common-блоки) Фортрана преобразованы в контейнеры данных на C++, а этапы обработки — в независимые задачи, что обеспечило модульность, сопровождаемость и сохранение физической достоверности результатов. Предложенная стратегия может быть тиражирована для модернизации другого унаследованного научного ПО.**

***Ключевые слова* — Baikal-GVD, качество ПО, реинжиниринг, реконструкция ливней заряженных частиц, унаследованное ПО.**

I. Введение

A. Проблема унаследованного кода в научных проектах

Научное ПО часто развивается десятилетиями, переходя от одного поколения разработчиков к другому. Исходные языки и инструменты устаревают, документация скудеет. Код превращается в "черный ящик": он работает (или кажется работающим), но понять его внутреннюю логику, модифицировать его или встроить его в современные конвейеры обработки становится крайне сложно. Это прямая угроза **сопровождаемости** и **функциональной пригодности** программы [1].

B. Исходный код реконструкции ливней на Фортране

В эксперименте Baikal-GVD алгоритм реконструкции ливней заряженных частиц был реализован в виде набора отдельных программ на Фортране. Каждая программа решала свою часть задачи (чтение данных, реконструкция времен, амплитуд и т.д.), и для получения конечного результата требовалось последовательно выполнить все эти программы, передавая данные через файлы. Все программы использовали одинаковые по структуре common-блоки, что впоследствии облегчило анализ — данные были организованы единообразно во всем коде.

C. Ключевые проблемы унаследованного кода

Дальнейшая эксплуатация исходного кода в рамках развивающейся инфраструктуры эксперимента оказалась затруднительной. Выделяются следующие проблемы.

- **Устаревшие и неподдерживаемые библиотеки.** Сборка фортранных компонентов в современном окружении стала затруднительной, а используемые математические библиотеки либо не развивались, либо были недоступны в новой инфраструктуре.
- **Сложность интеграции в современный фреймворк эксперимента.** Для Baikal-GVD таким фреймворком является BARS (Baikal Analysis and Reconstruction Software) [2], построенный на C++ и ROOT [3]. Вызов фортранного кода через обертки (wrappers) был возможен, но создавал узкие места: требовал дополнительных усилий на передачу данных между C++ и Фортраном и затруднял отладку, что неизбежно снижало общую **надежность** такой системы.
- **Сложность модификации и расширения.** В современных реалиях мало кто владеет Фортраном на уровне, позволяющем уверенно модифицировать такой код. Использование современных конфигурационных файлов или внедрение новой геометрии телескопа оказались неподъемными задачами. Это критически снижало **сопровождаемость** и **удобство использования**.

Перечисленные проблемы делали необходимой полную переработку кода, а не его поверхностную адаптацию.

D. Почему не автоматическая конвертация или перевод с помощью ИИ?

Варианты автоматической трансляции из Фортрана в C рассматривались. Инструменты автоматической трансляции, такие как f2c, обеспечивают лишь синтаксический перевод, но не решают задачу адаптации кода к современной архитектуре: результирующий код сохраняет все особенности исходной логики, включая неяв-

Статья получена 25 апреля 2026.

А.Г. Соловьев — старший научный сотрудник Лаборатории информационных технологий имени М.Г. Мещерякова, Объединенный Институт Ядерных Исследований; ORCID: https://orcid.org/0009-0001-9170-362X (e-mail: solovjev@jinr.ru)

Ж.-А.М. Джилкибаев — ведущий научный сотрудник Лаборатории нейтринной астрофизики высоких энергий, Институт ядерных исследований РАН (e-mail: djilkib@yandex.ru)

В.Я. Дик — младший научный сотрудник Лаборатории ядерных проблем им. В. П. Джелепова, Объединенный Институт Ядерных Исследований (e-mail: viktoriya@jinr.ru)

Т.В. Елжов — ведущий инженер Лаборатории ядерных проблем им. В. П. Джелепова, Объединенный Институт Ядерных Исследований (e-mail: telzhov@jinr.ru)

Б.А. Шайбонов — старший научный сотрудник Лаборатории ядерных проблем им. В. П. Джелепова, Объединенный Институт Ядерных Исследований (e-mail: bair@jinr.ru)

ные зависимости и глобальное состояние, а в ряде случаев становится даже менее читаемым. Использование больших языковых моделей (LLM) также не решает проблему — генерируемый код может быть синтаксически аккуратнее, однако требует тщательной верификации из-за риска скрытых ошибок (галлюцинаций). Кроме того, любой автоматический перевод сохраняет зависимость от фортранных библиотек времени выполнения.

E. Постановка задачи

Таким образом, требовалось не просто переписать код с Фортрана на C++, а создать современный, **сопровождаемый** компонент, который органично вписывается в архитектуру BARS, **понятен** новым разработчикам и допускает простое расширение. При этом необходимо было сохранить **физическую достоверность** результатов, накопленную за годы использования исходного кода.

II. Контекст: архитектура BARS

Для понимания дальнейшего изложения необходимо кратко описать базовую архитектуру программного комплекса, в который встраивается компонент.

BARS содержит набор программ, каждая из которых решает небольшую конкретную задачу. Вместе они реализуют все этапы обработки данных эксперимента Baikal-GVD.

Средства фреймворка позволяют создавать такие программы. В его основе лежат две фундаментальные абстракции:

- **контейнеры** (`MParContainer`) — хранят экспериментальные данные и параметры обработки;
- **задачи** (`MTask`) — преобразуют данные контейнеров, представляя элементарные шаги обработки.

На этой основе каждая программа организована в три этапа, каждый из которых включает проход по всем задачам из определенного пользователем списка `MTaskList`. На этапе предварительной обработки задачи в методе `PreProcess` регистрируют контейнеры и загружают параметры. Затем следует конвейер, организованный в виде **цикла событий** (рисунок 1): для каждого события задачи из списка `MTaskList` выполняются последовательно — для каждой задачи вызывается метод `Process`. Обмен данными между задачами осуществляется через общий список контейнеров `MParList`: каждая задача считывает входные данные из одних контейнеров и записывает результаты в другие. На заключительном этапе задачи в методе `PostProcess` финализируют результаты и освобождают ресурсы.

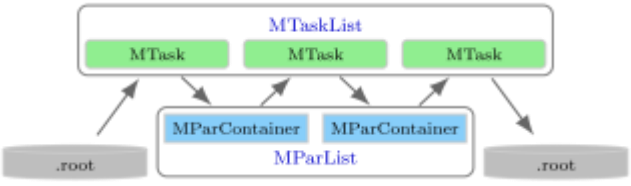

Рис. 1. Цикл обработки событий в BARS.

Проектируемый компонент — программа реконструкции ливней `reco-shower` — следует той же модели: конвейер из задач `MTask`, организованный в виде цикла событий. Это позволяет ей органично вписаться в архитектуру BARS и использовать те же принципы, что и остальные программы комплекса.

III. Стратегия реинжиниринга

A. Common-блоки Фортрана как основа для структур данных в C++

Общие блоки (common-блоки) Фортрана — это, по сути, слепок глобального состояния программы. Эти блоки стали естественной спецификацией и послужили отправной точкой для проектирования структур данных. Каждый common-блок был переработан и включён в тот или иной класс C++, унаследованный от `MParContainer` — стандартного контейнера данных в архитектуре BARS. Это позволило:

- гарантировать, что ни одно поле не потеряется при переводе;
- органично вписать новые классы C++ в архитектуру BARS, где задачи `MTask` обмениваются данными именно через контейнеры `MParContainer`.

B. Поэтапная замена и построение конвейера

Переписывание выполнялось последовательно по логике обработки данных: от чтения входных файлов к финальной записи. Каждый этап (`ReadRaw`, `Trec`, `A0`, `Ahit` и т.д.) реализовывался на C++ как отдельная задача `MTask` и по мере готовности добавлялся в список задач `MTaskList`, определяющий порядок обработки.

Для верификации каждого этапа была реализована опция вывода результатов в формате, идентичном исходной фортранной реализации. Это позволило на тестовых данных сравнивать результат каждого этапа с результатом соответствующей программы на Фортране.

Такой подход обеспечил:

- **корректность** — подтверждено, что результаты нового и старого кода совпадают;
- **надежность** — ошибки локализовались в пределах одного этапа;
- прозрачность и контролируемость процесса перевода.

Валидация реализации на C++ проводилась на тех же входных данных и калибровках, что и для фортранной версии. Сравнение результатов для характерных ранов показало, что подавляющее большинство событий восстанавливается обеими версиями идентично. Единичные расхождения (на уровне 1–2 событий на несколько миллионов входных событий) локализованы исключительно на этапе `Ahit` и обусловлены статистическими флуктуациями генератора случайных чисел RANLUX. Данные расхождения приемлемы как несущественные для физического анализа.

IV. Этапы конвейера reco-shower

Этот раздел выполняет роль документации компонента для разработчика, которому предстоит сопровождать или модифицировать код.

Методика реконструкции ливней была разработана для телескопа НТ-200 — первого глубоководного нейтринного телескопа на озере Байкал — и подробно описана в работах [4, 5, 6].

Программа `reco-shower` построена как последовательность задач `MTask`, выполняемых в цикле событий в следующем порядке:

- **`ReadRaw`** — чтение сырых данных, применение калибровок;
- **`RawAnalysis`** — первичный анализ, фильтрация шумовых срабатываний;
- **`Trec`** — реконструкция координат ливня по временам прихода сигналов;
- **`A0`** — грубая оценка энергии и направления ливня методом максимального правдоподобия;
- **`Ahit`** — проверка согласия модели с данными, оценка качества методом Монте-Карло;
- **`A0Fine`** — уточненная реконструкция энергии и направления на мелкой сетке;
- **`AhitFine`** — финальная проверка качества для уточненных параметров;
- **`WriteRootFile`** — запись всех восстановленных параметров в ROOT-файл.

Перед этапом `RawAnalysis` также выполняются служебные задачи: чтение конфигурации (`BReadStaticConfig`) и геометрии установки (`Geometry::Acquire`), преобразование координат в физическую систему (`BZhMKebkal2Phys`).

Для каждого этапа ниже поясняется его назначение и реализация — ровно настолько, чтобы разработчик понимал логику кода и мог его модифицировать. Если не указано иное, выходные данные одного этапа служат входными данными для следующего.

A. `ReadRaw`

Назначение: чтение сырых данных из входного потока (контейнер `BExtractedImpulseTel`), применение калибровок и преобразование во внутренние структуры программы, соответствующие фортранным common-блокам.

Реализация: в методе `PreProcess` выполняется однократная загрузка калибровочных констант для всего файла. В методе `Process` для каждого события производятся:

- отбраковка импульсов с отрицательной амплитудой или временем менее 400 нс;
- пересчет амплитуды в фотоэлектроны с использованием калибровочного коэффициента и проверка порога;
- расчет калиброванного времени с учетом индивидуального сдвига канала, задержки сигнала в кабельной системе и амплитудно-зависимой поправки;
- заполнение массивов для всех импульсов каждого канала.

Выходные данные: контейнер `BZhMDataReadRaw`, содержащий времена импульсов (`t`), амплитуды (`q`), максимальная амплитуда (`qmax`), соответствующее ей время (`tmax`), число импульсов на канал (`jpul`).

B. `RawAnalysis`

Назначение: фильтрация импульсов, полученных на этапе `ReadRaw`, отбраковка шумовых срабатываний и формирование компактного представления события для передачи в `Trec`.

Реализация: обработка выполняется в методе `Process` для каждого события в несколько этапов:

- по каналам с `qmax` $\geq$ `3` вычисляется средневзвешенное время `tav`; импульсы вне окна `tav` $\pm$ `500 ns` или с амплитудой ниже порога `athr = 1.5` отбрасываются;
- для каждого канала фиксируются первый по времени импульс и импульс с максимальной амплитудой;
- на основе этих данных формируется предварительный список сработавших каналов;
- рекурсивная процедура с использованием геометрии установки проверяет временную согласованность всех пар каналов с учетом скорости света; каналы с большими рассогласованиями последовательно исключаются;
- событие проходит отбор, если после фильтрации осталось не менее 7 каналов.

Выходные данные: контейнер `BZhMDataRawAnalysis`, содержащий времена (`tre`), амплитуды (`are`), число каналов (`jchw`), номера каналов (`npmtw`).

C. `Trec`

Назначение: восстановление пространственных координат ливня.

Реализация: процедура реконструкции выполняется в методе `Process` для каждого события итеративно с последовательной отбраковкой каналов:

- начальным приближением служат координаты канала с максимальной амплитудой;
- координаты ливня восстанавливаются минимизацией χ^2 между измеренными временами и ожидаемыми от точечного источника черенковского излучения [4, с. 14]; для минимизации в исходной реализации на Фортране использовался пакет Minuit из библиотеки CERNLIB [7]; в текущей версии применен `ROOT::Math::Minimizer` с тем же алгоритмом;
- время рождения ливня вычисляется аналитически на каждой итерации как среднее отклонение измеренных времен от ожидаемых;
- после фита вычисляются остаточные отклонения для каждого канала; каналы с отклонением, превышающим порог, отбраковываются, и фит повторяется;
- событие проходит отбор, если после всех итераций осталось не менее 5 каналов и не менее 3 гирлянд.

Выходные данные: контейнер `BZhMDataTrec`, содержащий координаты ливня (`oxt`), значение χ^2 (`ocht`), список каналов, прошедших все итерации (`ncurt`).

D. *A0 и A0Fine*

Назначение: восстановление энергии ливня и направления его оси.

Реализация: на вход подаются координаты ливня (результат `Trec`) и отфильтрованные амплитуды со списком каналов (результат `RawAnalysis`). В методе `PreProcess` выполняется однократная загрузка таблицы отклика оптических модулей на черенковское излучение ливня. В методе `Process` для каждого события выполняется сканирование по трехмерной сетке параметров (логарифм энергии, косинус зенитного угла, азимутальный угол):

- координаты оптических модулей пересчитываются в систему, связанную с текущим направлением ливня;
- для каждой точки сетки по таблице отклика определяется ожидаемое число фотонов для каждого канала и вычисляется логарифм функции правдоподобия (см. [4, с. 14]) на основе измеренных амплитуд `are`;
- точка сетки, минимизирующая $-2\ln L$, принимается за восстановленные параметры ливня.

Используются два режима работы:

- **`A0`** — грубое сканирование с широким шагом для предварительной оценки;
- **`A0Fine`** — уточняющее сканирование с мелким шагом в области, найденной на этапе `A0`.

Выходные данные: контейнеры `BZhMDataA0` или `BZhMDataA0Fine` содержащие энергию (`oxa[0]`), направление (`oxa[1]`, `oxa[2]`), значение функции правдоподобия (`ocha`).

E. *Ahit и AhitFine*

Назначение: оценка качества восстановления параметров ливня путем сравнения ожидаемого (по таблице отклика) и фактического числа сработавших каналов.

Реализация: в методе `PreProcess` загружается маска неработающих каналов и информация о статусе секций. В методе `Process` для каждого события:

- для каждого канала по таблице отклика определяется ожидаемое число фотонов `esn` с учетом геометрии и ориентации, вычисляется вероятность срабатывания $p = 1 - e^{-esn}$;
- вычисляется логарифм правдоподобия: для сработавших каналов вклад $\ln p$, для несработавших — аппроксимация $-esn$;
- методом Монте-Карло (100 испытаний) моделируются статистические флуктуации отклика каналов: для каждого канала разыгрывается факт срабатывания согласно вычисленной вероятности, и набирается суммарное значение функции правдоподобия; в исходной фортранной реализации использовался генератор RANLUX, в текущей — `TRandom1` с тем же алгоритмом;
- по результатам испытаний определяются среднее, минимальное и максимальное значения функции правдоподобия, используемые для отбора событий.

Используются два режима:

- **`Ahit`** — работает с результатами грубого сканирования `A0`;
- **`AhitFine`** — работает с результатами точного сканирования `A0Fine`.

Выходные данные: контейнеры `BZhMDataAhit` или `BZhMDataAhitFine` содержащие среднее (`ohnh`), минимальное (`ohmi`) и максимальное (`ohma`) значения функции правдоподобия; для каждого канала — ожидаемое число фотонов (`oa`) и разность между измеренной и ожидаемой амплитудой (`oda`).

F. *WriteRootFile*

Назначение: запись всех восстановленных параметров события и дополнительных расчетных величин в файл в формате ROOT для последующего физического анализа.

Реализация: в методе `PreProcess` создаётся файл с деревом `TTree`, ветви которого соответствуют выходным данным всех предыдущих этапов. В методе `Process` для каждого события:

- координаты оптических модулей пересчитываются в систему координат ливня;
- выполняется классификация импульсов: сравнение времён всех зарегистрированных импульсов с ожидаемыми временами от ливня, мюона и фона;
- рассчитываются ожидаемые времена прихода мюонного и фонового сигналов;
- с использованием функций из библиотеки SLALIB [8] вычисляются экваториальные координаты события: прямое восхождение и склонение;
- из файла, указанного в конфигурации, загружается список астрофизических источников-кандидатов; для каждого источника вычисляется угловое расстояние между направлением события и направлением на источник;
- все параметры записываются в дерево.

В методе `PostProcess` файл сохраняется на диск.

Сохраняемые данные: результаты `RawAnalysis` (времена `tre`, амплитуды `are`), `Trec` (координаты `oxt`, качество фита `ocht`, остаточные отклонения `odt`), `A0`/`A0Fine` (энергия и направление `oxa`, правдоподобие `ocha`), `Ahit`/`AhitFine` (критерии качества `ohnh`, `ohmi`, `ohma`, ожидаемые амплитуды `oa`, разности `oda`).

Дополнительные выходные данные: координаты модулей в системе ливня (`xarn`, `yarn`, `zarn`), классификация импульсов (`ntype[0..3]`), ожидаемые времена мюонного (`tmu`) и фонового (`tbg`) сигналов, экваториальные координаты события, угловые расстояния до астрофизических источников.

V. Заключение

A. Модернизированный компонент реконструкции ливней в свете стандартов качества ПО

Проделанная работа позволила не просто переписать код с Фортрана на C++, но и существенно повысить качество компонента по ключевым характеристикам [1].

- **Сопровождаемость**. В терминах характеристик качества это выражается в:

- **модульности** — каждый этап обработки оформлен как отдельная задача `MTask` в цикле событий BARS, что позволяет править их независимо;
- **модифицируемости** — расширение функциональности (например, поддержка новой геометрии телескопа) не требует переписывания всего кода;
- **анализируемости** — код хорошо структурирован, читаем (написан на современном C++) и легко поддаётся анализу стандартными инструментами разработки;
- **тестируемости** — наличие отладочного режима вывода, идентичного фортранной реализации, позволяет проверять корректность каждого этапа.

- **Удобство использования.** В терминах характеристик качества это выражается в:
  - **определимости пригодности** — программа `reco-shower` следует стандартным для фреймворка соглашениям, что делает ее назначение и использование очевидными для любого пользователя, работающего с BARS;
  - **изучаемости** — для работы не требуется знание Фортрана, а настоящая статья служит документацией.
- **Функциональная пригодность** сохранена. В терминах характеристик качества это выражается в:
  - **функциональной полноте** — реализованы все этапы обработки исходной реализации на Фортране (от `ReadRaw` до `WriteRootFile`);
  - **функциональной корректности** — сохранена физическая достоверность результатов, накопленная за годы использования исходного кода; новый код считает не хуже старого.
- **Уровень производительности.** Хотя оптимизация не являлась основной целью работы, удалось улучшить и **временные характеристики**:
  - исключен промежуточный файловый ввод-вывод — данные передаются между этапами напрямую через контейнеры в памяти, что заведомо исключает накладные расходы на дисковые операции, присутствовавшие в исходной реализации;
  - реализация в виде независимых задач `MTask` создает основу для многопоточной обработки цикла событий [9].

B. Извлеченные уроки

Помимо работающего компонента реконструкции, результатом проделанной работы стали уроки, касающиеся подхода к унаследованному коду в целом.

- **Common-блоки как карта данных.** Глобальные области памяти в Фортране, которые обычно воспринимаются как устаревшее наследие, на самом деле стали отправной точкой для проектирования структур данных. Они дали готовую спецификацию того, с чем работает алгоритм, и позволили ничего не упустить при переводе.
- **Унаследованный код как инструмент достижения качества ПО.** Наличие исходной фортранной реализации обеспечило:
  - **корректность реализации** — исходный код послужил детальной спецификацией алгоритмов, что позволило воспроизвести их без искажений (**функциональная корректность**);
  - **основу для тестирования** — *работающая* исходная реализация предоставила естественный эталон для проверки корректности и позволила организовать сравнение результатов без разработки сложных тестовых сценариев (**тестируемость**).

Опыт модернизации реконструкции ливней является примером реализации принципов устойчивого научного ПО, сформулированных в нашей работе [10]. Анализ унаследованных решений обеспечил функциональную пригодность новой реализации, а сопровождение разработки научной публикацией — ее удобство использования и долгосрочную сопровождаемость.

C. Значение для дальнейших работ

Описанный подход — не разовое решение, а воспроизводимая стратегия. Она уже была применена к другому компоненту: чтение данных Монте-Карло (`MCRead`) также было переработано и встроено в конвейер BARS по той же методике. Ключевые элементы стратегии — анализ common-блоков, поэтапная замена с верификацией, встраивание в современный конвейер — инвариантны и переносимы.

Благодарности

Авторы выражают признательность коллаборации Baikal-GVD за предоставление материалов и условий для работы.

Библиография

[1] ГОСТ Р ИСО/МЭК 25010-2015. Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов. — М.: Стандартинформ, 2015.

[2] Avrorin A.D. et al. (Baikal-GVD Collaboration). Automatic data processing for Baikal-GVD neutrino observatory // Proceedings of Science. — 2021. — Vol. ICRC2021. — P. 1040. — DOI: 10.22323/1.395.1040. — arXiv:2107.13939.

[3] Brun R., Rademakers F. ROOT — An object oriented data analysis framework // Nuclear Instruments and Methods in Physics Research A. — 1997. — Vol. 389, Issues 1-2. — P. 81-86. DOI: 10.1016/S0168-9002(97)00048-X

[4] Шайбонов Б.А. Выделение событий от каскадов, инициированных мюонами и нейтрино, в экспериментах на Байкальском глубоководном нейтринном телескопе: автореф. дис. ... канд. физ.-мат. наук. — М.: Институт ядерных исследований РАН, 2010. — 29 с.

[5] Avrorin A.V. et al. (Baikal-GVD Collaboration). Search for a diffuse flux of high-energy extraterrestrial neutrinos with the NT-200 neutri-

no telescope // Astroparticle Physics. — 2006. — Vol. 25. — P. 140–150.

[6] Avrorin A.V. et al. (Baikal-GVD Collaboration). Search for high-energy neutrinos in the Baikal neutrino experiment // Astronomy Letters. — 2009. — Vol. 35, No. 10. — P. 651–662.

[7] CERN Program Library Long Writeups. — Geneva: CERN, 1993. — URL: https://cernlib.web.cern.ch

[8] Wallace P.T. The SLALIB Library // Astronomical Data Analysis Software and Systems III, ASP Conference Series, Vol. 61. — 1994. — P. 481.

[9] Соловьев А.Г. и др. (Коллаборация Baikal-GVD). Многопоточность для базового программного фреймворка Baikal-GVD // Физика элементарных частиц и атомного ядра. — 2026. — Т. 57, № 4, с. 598–602.

[10] Соловьев А.Г. и др. Принципы устойчивого научного ПО: опыт разработки программы обработки данных малоуглового рассеяния нейтронов // Компьютерные исследования и моделирование. — 2026. — Т. 18, № 2, с. 335–358.

Reengineering of the cascade reconstruction in the Baikal-GVD experiment: principles of working with legacy scientific software

A.G. Solovjev, Zh.-A.M. Dzhilkibaev, V.Y. Dik, T.V. Elzhov, B.A. Shaybonov

***Abstract* — This paper describes the reengineering of legacy Fortran code for charged-particle cascade reconstruction in the Baikal-GVD experiment and its integration into the modern C++ BARS framework. Instead of automatic conversion, a manual step-by-step translation was employed, with verification at each stage. Fortran common blocks were transformed into C++ data containers, and processing stages were converted into independent tasks, ensuring modularity, maintainability, and preservation of the physical validity of the results. The proposed strategy can be replicated for the modernization of other legacy scientific software.**

***Keywords* — Baikal-GVD, software quality, reengineering, charged-particle cascade reconstruction, legacy scientific software.**

REFERENCES

[1] GOST R ISO/IEC 25010-2015. Information technology. Systems and software engineering. Systems and software quality requirements and evaluation (SQuaRE). System and software quality models. Moscow: Standartinform, 2015. (in Russian)

[2] Avrorin A.D. et al. (Baikal-GVD Collaboration). Automatic data processing for Baikal-GVD neutrino observatory // Proceedings of Science. — 2021. — Vol. ICRC2021. — P. 1040. — DOI: 10.22323/1.395.1040. — arXiv:2107.13939.

[3] Brun R., Rademakers F. ROOT — An object oriented data analysis framework // Nuclear Instruments and Methods in Physics Research A. — 1997. — Vol. 389, Issues 1-2. — P. 81-86. DOI: 10.1016/S0168-9002(97)00048-X

[4] Shaybonov B.A. Selection of events from cascades initiated by muons and neutrinos in experiments with the Baikal deep-water neutrino telescope. PhD thesis abstract. Moscow: Institute for Nuclear Research RAS, 2010. 29 p. (in Russian)

[5] Avrorin A.V. et al. (Baikal-GVD Collaboration). Search for a diffuse flux of high-energy extraterrestrial neutrinos with the NT-200 neutrino telescope // Astroparticle Physics. — 2006. — Vol. 25. — P. 140–150.

[6] Avrorin A.V. et al. (Baikal-GVD Collaboration). Search for high-energy neutrinos in the Baikal neutrino experiment // Astronomy Letters. — 2009. — Vol. 35, No. 10. — P. 651–662.

[7] CERN Program Library Long Writeups. — Geneva: CERN, 1993. — URL: https://cernlib.web.cern.ch

[8] Wallace P.T. The SLALIB Library // Astronomical Data Analysis Software and Systems III, ASP Conference Series, Vol. 61. — 1994. — P. 481.

[9] Soloviev A.G. et al. (Baikal-GVD Collaboration). Multi-Threading for Baikal-GVD Core Software Framework. Phys. Part. Nuclei 57, 609–611 (2026). — DOI: 10.1134/S1063779626700279

[10] Soloviev A.G. et al. Principles of sustainable scientific software: lessons from developing a data processing program for small-angle neutron scattering // Computer Research and Modeling, 2026, vol. 18, no. 2, pp. 335-358. (in Russian)