Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

# Реализация поддержки Qt 5.15 в генераторе библиотеки PythonQt

***Уфимцев Юрий Дмитриевич***

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
старший преподаватель кафедры СП Я. А. Кириленко

Санкт-Петербург
2024

# Оглавление

**Введение** **3**

**1. Постановка задачи** **4**

**2. Обзор** **5**

2.1. Современные способы и средства тестирования . . . . . 5

2.2. Утечки памяти . . . . . 6

2.3. Состояние проекта . . . . . 7

2.4. Используемые технологии . . . . . 8

**3. Реализация** **9**

3.1. Поддержка Qt 5.12 . . . . . 9

3.2. Тестирование качества работы генератора . . . . . 10

3.3. Анализ и устранение утечек памяти . . . . . 10

3.4. Обнаружение и устранение ошибок . . . . . 12

**4. Апробация** **13**

**Заключение** **14**

**Список литературы** **15**

# Введение

Взаимодействие кода на C++ и Python является важной задачей для различных проектов: платформа обработки медицинских изображений и научной визуализации MeVisLab [18], среда программирования роботов TRIK Studio [20]. TRIK Studio, например, позволяет писать скрипты на Python и организовывает взаимодействие написанного кода с API контроллера TRIK, реализованном на языке C++ с использованием библиотеки Qt [21].

Для решения нетривиальной задачи вызова кода на C++ из Python существует open-source библиотека PythonQt [11], которая содержит функции-обертки для методов модулей библиотеки Qt, созданные так, чтобы они могли быть динамически загружены, распознаны и корректно вызваны кодом на Python.

Важным компонентом библиотеки PythonQt является генератор, ответственный за обработку кода Qt и создание соответствующих функций-обёрток. Фреймворк Qt постоянно развивается и активно поддерживается, внедряя новые конструкции и стандарты языка C++. Однако, генератор развивается более медленными темпами по сравнению с Qt, из-за чего в репозитории PythonQt в момент начала работы находились сгенерированные функции-обертки лишь для модулей Qt версии 5.11. Для кода модулей Qt 5.15 и старше генератор не мог распознать все конструкции языка C++ и сгенерировать нужные файлы; для версий Qt 5.12 и младше мог, но с набором ошибок и предупреждений.

Таким образом, существовала проблема миграции на Qt 5.15 и Qt 6. Эта проблема затрагивала множество пользователей, не позволяя перестать использовать версии Qt, поддержка которых уже завершена.

# 1. Постановка задачи

Требуется поддержать работу генератора с Qt 5.12 и упростить процесс поддержки работы генератора с версиями Qt 5.15 и старше с помощью повышения качества работы генератора и реализации механизма тестирования.

Для достижения цели были поставлены следующие задачи:

1. Поддержать Qt 5.12 и C++ 11 в генераторе библиотеки PythonQt.
2. Предложить механизм тестирования качества работы генератора.
3. Устранить утечки памяти генератора.
4. Реализовать механизм тестирования качества работы генератора. Избавиться от найденных ошибок.

# 2. Обзор

Данная работа является достаточно разноплановой: используется множество различных понятий, технологий и методов решения задач. Для лучшего понимания работы обзор состоит из исследования современных способов и средств тестирования, отладки утечек памяти, включает описание состояния проекта на момент начала работы и анализ используемых технологий.

## 2.1. Современные способы и средства тестирования

В процессе тестирования важными компонентами являются непрерывная интеграция (Continuous Integration) и санитайзеры.

### 2.1.1. CI

Непрерывная интеграция — это практика разработки программного обеспечения, при которой изменения в коде интегрируются в общий репозиторий достаточно часто, а также автоматически проходят цикл сборки и тестирования в рабочих потоках.

Рабочий поток (workflow) — основа CI, представляющая собой серию автоматизированных шагов, выполняемых при интеграции кода в общий репозиторий. Эти шаги включают в себя сборку, тестирование и другие задачи, например, развертывание приложения.

Важной частью тестирования является проверка программы на неопределенное поведение и утечки памяти.

### 2.1.2. Санитайзеры

Неопределенное поведедение (Undefined Behaviour) [10], в отличие от наблюдаемого и поведения, определенного реализацией, характеризуется отсутствием ограничений на работу программы. Примерами неопределенного поведения являются гонки данных, обращения к памяти за пределами границ массива, переполнение знакового целого числа, разы-

менование нулевого указателя, доступ к объекту через указатель другого типа.

Для выявления дефектов программы, приводящих к неопределенному поведению, а также к некорректной работе с памятью, к появлению гонок данных и взаимоблокировок используются санитайзеры — инструменты, которые обнаруживают ошибки во время выполнения с помощью добавления компилятором инструментального кода.

Основные виды санитайзеров [6]:

- AddressSanitizer (ASan) [3] используется для обнаружение ошибок работы с памятью, таких как выход за пределы массива и двойное освобождение памяти. В состав AddressSanitizer входит LeakSanitizer [4], обнаруживающий утечки памяти.

- UndefinedBehaviorSanitizer (UBSan) [8] предназначен для обнаружения неопределенного поведения во время выполнения программы.

- ThreadSanitizer (TSan) [7] используется для обнаружения гонок данных в многопоточных программах.

- MemorySanitizer (MSan) [5] предназначен для обнаружения обращений к неинициализированным областям памяти.

Поскольку в программе не используется многопоточность, были выбраны UBSan в качестве важнейшего средства регрессионного тестирования и ASan для обнаружения ошибок работы с памятью, включая LeakSanitizer для обнаружения утечек.

## 2.2. Утечки памяти

Утечка памяти (memory leak) — процесс неконтролируемого уменьшения свободной виртуальной памяти, характеризующийся тем, что приложение выделяет блок памяти, но забывает освободить его. Утечки приводят к избыточному использованию ресурсов, снижению произ-

водительности приложения. При значительном уменьшении свободной памяти приложение может достичь лимита доступной памяти.

Устранение утечек памяти включает в себя динамическую отладку с использованием вспомогательных инструментов:

- модуль memcheck инструмента отладки и профилирования Valgrind [14];
- LeakSanitizer в составе AddressSanitizer;
- Visual Leak Detector [19];
- heaptrack [12].

В качестве основного инструмента для отладки утечек памяти был выбран LeakSanitizer, поскольку он поставлялся в рамках AddressSanitizer и достаточно быстро генерировал информацию об утечках. В качестве более мощного средства обнаружения утечек памяти для проверки результатов LeakSanitizer использовался Valgrind. Этот инструмент работал дольше, но генерировал более структурированный отчет об утечках. Также Valgrind был доступен в QtCreator [9] и не требовал дополнительной установки, в отличие от Visual Leak Detector и heaptrack.

## 2.3. Состояние проекта

На момент начала работы в репозитории PythonQt существовал workflow «Build», выполняющий сборку и запуск генератора, сборку PythonQt со вспомогательными файлами, расположенными в репозитории, и запуск тестов.

В этом рабочем потоке отсутствовало регрессионное и интеграционное тестирование модулей библиотеки, направленное на обнаружение ошибок в выпущенных генератором файлах и проверку взаимодействия генератора и среды выполнения библиотеки PythonQt.

Поэтому в рамках данной учебной практики было необходимо добавить регрессионное ингтеграционное тестирование, включая провер-

ку на неопределенное поведение и нефункциональное тестирование на утечки памяти с помощью современных технологий.

## 2.4. Используемые технологии

Во время работы над генератором библиотеки PythonQt использовались:

- среда разработки QtCreator для написания кода на языке C++ с использованием модулей библиотеки Qt;
- язык C++;
- модули библиотеки Qt версий 5.12, 5.15;
- компилятор g++ языка C++, поставляемый в пакете GNU Compiler Collection версии 11.4.0 [2];
- санитайзеры AddressSanitizer, LeakSanitizer и UndefinedBehaviorSanitizer для анализа утечек памяти и выявления ошибок работы с памятью, неопределенного поведения генератора;
- модуль memcheck набора инструментов Valgrind для выявления проблем при работе с памятью и обнаружения утечек памяти;
- программа Meld [13] для сравнения содержимого выпущенных генератором файлов.

# 3. Реализация

Работа проходила в несколько этапов: поддержка Qt 5.12 в генераторе библиотеки PythonQt; предложение и реализация механизма тестирования качества работы генератора; анализ и устранение утечек памяти; обнаружение и устранение ошибок.

## 3.1. Поддержка Qt 5.12

Запуск генератора на модулях библиотеки Qt 5.12 происходил с набором предупреждений двух видов: «WARNING» и «SHIT». Предупреждения с меткой «SHIT» привлекали к себе внимание и вызывали сомнения относительно корректности функционирования генератора. Вследствие этого было принято решение найти и устранить причину появления предупреждений второго типа.

Также у генератора отсутствовал механизм сообщения о предупреждениях с указанием адреса в коде, включающего номер строки. Для повышения эффективности и удобства процесса отладки и разработки был создан новый механизм информирования о предупреждениях.

После анализа кода модулей библиотеки Qt было установлено, что предупреждения возникают в результате отсутствия поддержки генератором enumeration-объявлений стандарта C++ 11 вида «enum class» [1]. В код парсера, входящего в состав генератора, были внесены изменения, позволяющие корректно обрабатывать подобные конструкции стандарта C++ 11.

С помощью сравнения кода, полученного в результате работы генератора, с предыдущей версией было обнаружено, что благодаря корректному распознаванию конструкции C++ 11 были добавлены новые записи. Генерация изменилась в положительную сторону.

Также был выявлен набор предупреждений, связанный с перечислениями: в файлах вида typesystem.xml, содержащих информацию о типах и структурах данных, доступных при взаимодействии Python и C++, не были указаны некоторые перечисления. В соответствующие места были добавлены записи о перечислениях, распознанных после

внесения изменений в код, и перечислениях, записи о которых по неизвестным причинам не были добавлены ранее.

## 3.2. Тестирование качества работы генератора

Как ранее было указано в обзоре, возникла необходимость в добавлении шага регрессионного и интеграционного тестирования генератора. Требовалось:

- выполнять сборку PythonQt с использованием только что сгенерированных файлов, а не уже хранящихся в репозитории;
- использовать UBSan, ASan и LSan для проверки качества работы генератора;

Был создан новый workflow «Check generated cpp», осуществляющий интеграционное и регрессионное тестирование генератора и среды выполнения библиотеки PythonQt на версиях Qt 5.12, 5.15 и Qt 6: сборка среды выполнения PythonQt выполнялась на основе результатов, полученных от генератора на предыдущем шаге. Были добавлены санитайзеры UBSan и ASan, LSan был отключен из-за обнаруженных 58 МБ утечек памяти генератора. Для проверки регресса генератора после внесенных изменений в его код добавление LSan было необходимо, поэтому потребовалось устранить утечки памяти.

## 3.3. Анализ и устранение утечек памяти

Были выделены два способа устранения утечек памяти:

1. Явное освобождение динамически выделенной памяти в местах утечек, включая деструкторы объектов, с помощью оператора delete [17] и метода qDeleteAll [22].
2. Использование умных указателей QSharedPointer [16], управляющих временем жизни удерживаемого объекта. В отличие от обычных указателей, QSharedPointer реализует автоматический счет-

чик ссылок и освобождает память, когда последняя ссылка на объект уничтожается.

В коде генератора библиотеки PythonQt умные указатели не использовались, поэтому чтобы не влиять на архитектуру проекта, было принято решение использовать первый способ. Был проанализирован стек вызовов набора утечек и были добавлены соответствующие методы освобождения памяти. Объем утекаемой памяти уменьшился с 58 МБ до 28 МБ.

Оставшиеся утечки не могли быть устранены с помощью первого способа из-за архитектурных особенностей проекта. В частности, была обнаружена проблема владения объектом, вызванная:

- добавлением в одну коллекцию указателей на один и тот же объект в памяти;
- добавлением в разные коллекции указателей на один и тот же объект в памяти.

Были выделены два возможных решения:

1. Создание методов глубокого копирования для добавления в коллекцию копий объектов.
2. Замена указателей на данные объекты умными указателями.

Первый вариант предполагал реализацию методов глубокого копирования для множества объектов с большим количеством полей. Это влекло за собой необходимость добавления сотен строк кода и возможность возникновения новых ошибок при копировании объектов. В связи с этим было принято решение заменить указатели на умные указатели. Такой подход позволял избежать возможных дублирований объектов и упрощал процесс управления памятью.

Стоит отметить, что требующие замены указатели в PythonQt были реализованы с помощью класса CodeModelPointer, наследника QAtomicPointer [15]. Код генератора PythonQt содержал набор методов, обеспечивающих корректную работу с подобными указателями:

например, методы статического и динамического приведения объекта, удерживаемого указателем, к другому типу.

Были внесены изменения по замене CodeModelPointer на QSharedPointer, необходимые для осуществления своевременного освобождения памяти и корректной работы генератора. Результатом стала практически полная ликвидация утечек памяти: объем уменьшился с 28 МБ до 300 КБ.

Причиной оставшихся утечек также являлась проблема владения, связанная с объектами типа TypeEntry. Поскольку этот класс имел много наследников, было принято решение отказаться от замены указателя на QSharedPointer и не вводить методы глубокого копирования: внесение необходимых изменений в код оказалось слишком затратным для устранения относительно небольшого объема утечек. Эти утечки было решено добавить в файл игнорирования LeakSanitizer.

## 3.4. Обнаружение и устранение ошибок

В процессе выполнения задач:

- была выявлена и рассмотрена проблема нестабильной генерации, влияющая на проверку результатов внесенных в генератор изменений;
- были обнаружены и исправлены такие ошибки, как выход за границу массива из-за недостаточной синхронизации двух объектов, отсутствие копирования удерживаемого указателем объекта при возвращении из функции.

Также в ходе работы велись дискуссии с другими разработчиками, обсуждались текущие задачи и методы их решения.

# 4. Апробация

Проверка корректности предлагаемых изменений обеспечивалась следующими действиями:

- локальная сборка и запуск генератора, сборка PythonQt и запуск тестов;
- использование санитайзеров ASan и UBSan, запускаемых с различными опциями для детального изучения и анализа;
- сравнение вновь сгенерированных файлов с предыдущими результатами с помощью программы Meld;
- запуск CI генератора в своем репозитории и в оригинальном репозитории PythonQt, включая стандартный workflow сборки и workflow, реализующий регрессионное и интеграционное тестирование генератора;
- ревью пул-реквестов сообществом разработчиков.

# Заключение

В результате были решены следующие задачи:

1. Поддержаны Qt 5.12 и C++ 11 в генераторе библиотеки PythonQt.
2. Предложен механизм тестирования качества работы генератора.
3. Устранены утечки памяти генератора.
4. Реализован механизм тестирования качества работы генератора. Устранены найденные ошибки.

Результаты работы можно найти в репозитории PythonQt: https://github.com/MeVisLab/pythonqt. Изменения вносились в следующих пул-реквестах:

- https://github.com/MeVisLab/pythonqt/pull/136;
- https://github.com/MeVisLab/pythonqt/pull/147;
- https://github.com/MeVisLab/pythonqt/pull/150;
- https://github.com/MeVisLab/pythonqt/pull/151;
- https://github.com/MeVisLab/pythonqt/pull/177;
- https://github.com/MeVisLab/pythonqt/pull/133.

# Список литературы


[1] Enumeration declaration. — URL: https://en.cppreference.com/w/cpp/language/enum (дата обращения: 2023-12-24).

[2] GNU Compiler Collection. — URL: https://gcc.gnu.org/ (дата обращения: 2023-12-24).

[3] Google. AddressSanitizer. — URL: https://github.com/google/sanitizers/wiki/AddressSanitizer (дата обращения: 2023-12-24).

[4] Google. LeakSanitizer. — URL: https://github.com/google/sanitizers/wiki/AddressSanitizerLeakSanitizer (дата обращения: 2023-12-24).

[5] Google. MemorySanitizer. — URL: https://github.com/google/sanitizers/wiki/MemorySanitizer (дата обращения: 2023-12-24).

[6] Google. Sanitizers. — URL: https://github.com/google/sanitizers (дата обращения: 2023-12-24).

[7] Google. ThreadSanitizer. — URL: https://github.com/google/sanitizers/wiki/ThreadSanitizerCppManual (дата обращения: 2023-12-24).

[8] Google. UndefinedBehaviourSanitizer. — URL: https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html (дата обращения: 2023-12-24).

[9] QtCreator. — URL: https://doc.qt.io/qtcreator/ (дата обращения: 2023-12-24).

[10] Undefined behavior. — URL: https://en.cppreference.com/w/cpp/language/ub (дата обращения: 2023-12-24).

[11] Библиотека PythonQt. — URL: https://mevislab.github.io/pythonqt/ (дата обращения: 2023-12-24).

[12] Инструмент для профилирования памяти heaptrack. — URL: https://github.com/KDE/heaptrack (дата обращения: 2023-12-24).

[13] Инструмент для сравнения содержимого текстовых файлов или каталогов Meld. — URL: https://meldmerge.org/ (дата обращения: 2023-12-24).

[14] Инструментальное программное обеспечение Valgrind. — URL: https://valgrind.org/docs/manual/mc-manual.html (дата обращения: 2023-12-24).

[15] Класс QAtomicPointer. — URL: https://doc.qt.io/qt-5/qatomicpointer.html (дата обращения: 2023-12-24).

[16] Класс QSharedPointer. — URL: https://doc.qt.io/qt-5/qsharedpointer.html (дата обращения: 2023-12-24).

[17] Оператор delete. — URL: https://en.cppreference.com/w/cpp/memory/new/operator_delete (дата обращения: 2023-12-24).

[18] Платформа обработки медицинских изображений и научной визуализации MeVisLab. — URL: https://www.mevislab.de/ (дата обращения: 2023-12-24).

[19] Программа для поиска утечек памяти Visual Leak Detector. — URL: https://kinddragon.github.io/vld/ (дата обращения: 2023-12-24).

[20] Среда программирования роботов TRIK Studio. — URL: https://trikset.com/products/trik-studio (дата обращения: 2023-12-24).

[21] Фреймворк Qt для разработки на языке программирования C++. — URL: https://www.qt.io/ (дата обращения: 2023-12-24).

[22] Функция qDeleteAll. — URL: https://doc.qt.io/qt-6/qtalgorithms.html#qDeleteAll (дата обращения: 2023-12-24).