

ИССЛЕДОВАНИЕ И РЕАЛИЗАЦИЯ ПРОЦЕССА ОБЕСПЕЧЕНИЯ КАЧЕСТВА АППАРАТНО-ПРОГРАММНЫХ СРЕДСТВ КОМПЬЮТЕРНЫХ СИСТЕМ

Ц.С. Оганнисян, А.К. Галстян

Национальный политехнический университет Армении

Эффективная, безопасная и надежная работа систем в значительной степени зависит от процессов обеспечения качества. Обеспечение качества (Quality Assurance, QA) в аппаратно-программных средствах представляет собой совокупность организованных и систематических мероприятий, направленных на обеспечение соответствия как аппаратных, так и программных компонентов установленным требованиям, стандартам и ожиданиям пользователей на протяжении всего жизненного цикла системы.

Обеспечение качества в аппаратно-программных системах включает анализ требований, контроль проектирования, различные уровни тестирования, проверку интеграции, а также оценку поведения системы в реальных и моделируемых условиях. Поскольку эти системы состоят из взаимосвязанных аппаратных и программных компонентов, корректность и стабильность их взаимодействия имеют важное значение. Даже незначительные ошибки могут привести к отказу системы, потере данных или проблемам безопасности, что особенно критично для систем, применяемых в жизненно важных областях.

Аппаратно-программные решения интегрируют программные и физические компоненты, что обуславливает формирование более сложных требований к качеству по сравнению с традиционным программным обеспечением.

В условиях стремительного развития современных технологий ручное тестирование уже не является достаточным, что обусловлено рядом факторов, включая человеческий фактор, большой объем тестовых сценариев, разнообразие рабочих сред и требования к быстрому выпуску программных продуктов (Release). В связи с этим автоматизация процессов обеспечения качества стала не только целесообразной, но и обязательной предпосылкой для повышения эффективности и точности тестирования.

В данной работе рассматриваются основные теоретические принципы обеспечения качества аппаратно-программных систем, их значимость, а также вопросы проектирования и реализации автоматизированной системы обеспечения качества с использованием Python, Playwright и ряда инструментов DevOps.

Ключевые слова: обеспечение качества, GitLab, DevOps, Prometheus.

Введение. В условиях развития современных компьютерных систем обеспечение качества программных и аппаратных компонентов (Quality Assurance) представляет собой один из ключевых процессов, направленных на гарантирование надёжности, функциональности и устойчивости работы системы на протяжении всего жизненного цикла. Обеспечение качества играет важную роль в повышении надёжности систем, снижении рисков и обеспечении общей эффективности продукта. Оно позволяет выявлять и устранять дефекты на ранних этапах, снижать затраты на последующие исправления и обеспечивать соответствие системы стандартам качества. Кроме того, эффективная реализация процессов обеспечения качества повышает доверие пользователей к системе и способствует долгосрочной устойчивости и конкурентоспособности технологических решений. Таким образом, обеспечение качества в аппаратно-программных средствах является не только технической необходимостью, но и важным фактором обеспечения безопасной, надёжной и эффективной эксплуатации систем.

Постановка задачи. Целью работы является исследование, разработка и реализация автоматизированного механизма обеспечения качества, обеспечивающего высокую эффективность тестирования аппаратно-программных систем, автоматизированную генерацию отчётности, непрерывную интеграцию и мониторинг в режиме реального времени.

1. Понятие качества и его значение. В инженерной практике качество определяется как способность системы удовлетворять заранее установленным требованиям. Основной целью обеспечения качества является предотвращение возникновения ошибок в системе, тогда как тестирование направлено на выявление уже существующих дефектов.

В общем виде качество может быть представлено следующей формулой:

$$Q = \frac{S_{\text{соответствующий}}}{S_{\text{общий}}},$$

где Q - уровень качества; $S_{\text{соответствующий}}$ - количество тестов, соответствующих установленным требованиям; $S_{\text{общий}}$ - общее количество требований / тестовых случаев.

Данная формула носит упрощённый характер, однако наглядно отражает основную цель и принцип измерения контроля качества.

2. Распределение ролей в процессе обеспечения качества. Обеспечение качества (QA) включает в себя совокупность методов и практик, среди которых можно выделить следующие этапы:

- анализ требований;
- разработка тестовых сценариев;
- подготовка тестовых сред;
- ручное и автоматизированное тестирование;
- анализ результатов;
- мониторинг и непрерывное совершенствование.

3. Влияние человеческого фактора. Несмотря на профессиональный опыт специалистов по обеспечению качества, при ручном тестировании неизбежно возникает ряд рисков, к которым относятся:

- невнимательность;
- некорректная интерпретация требований и комментариев;
- ограниченность времени;
- сложность проверки больших объемов данных;
- утомляемость.

Данное явление может быть представлено в виде упрощенной формулы вероятности возникновения ошибки:

$$P_{\text{ош}} = P_{\text{чел}} + P_{\text{ср}} + P_{\text{сл}}$$

где $P_{\text{чел}}$ — человеческий фактор; $P_{\text{ср}}$ — влияние среды; $P_{\text{сл}}$ — случайные факторы.

Автоматизация напрямую снижает влияние человеческого фактора, делая процесс обеспечения качества более устойчивым, воспроизводимым и контролируемым.

Необходимость автоматизации. К основным преимуществам автоматизированных систем обеспечения качества относятся:

- высокая скорость выполнения тестов;
- воспроизводимость и предсказуемость результатов;
- снижение вероятности человеческих ошибок;
- возможность выполнения тестов большого объема;
- поддержка процессов непрерывной интеграции;
- ускорение CI/CD-цикла.

Общая эффективность процесса может быть представлена следующей формулой:

$$E = \frac{T_{\text{ручн}} - T_{\text{авт}}}{T_{\text{авт}}} \times 100\%,$$

где $T_{\text{ручн}}$ — продолжительность ручного тестирования; $T_{\text{авт}}$ — продолжительность автоматизированного тестового цикла.

Во многих практических случаях показатель эффективности превышает 70...90%.

Выбор используемых технологий и инструментальных средств.

Для обеспечения автоматизации прежде всего необходимо корректно выбрать инструментальную базу, позволяющую сделать систему максимально универсальной и практичной. Рассмотрим выбор языка программирования и рабочего фреймворка для автоматизации. В качестве основного языка программирования был выбран Python, что обусловлено следующими факторами:

- простой и лаконичный синтаксис, снижающий временные затраты на разработку, и простой и лаконичный синтаксис, снижающий временные затраты на разработку и сопровождение тестов;
- наличие обширной экосистемы с десятками тысяч библиотек, упрощающих интеграцию (API-тестирование, работа с базами данных, отчётность, CI/CD-процессы);
- согласно многочисленным отраслевым источникам, Python является одним из самых популярных языков автоматизации QA в мире, что облегчает командную работу и быстрое вовлечение новых специалистов;
- высокая степень интеграции с инструментами разработки, обеспечения качества и вспомогательными системами.

В качестве фреймворка для автоматизации был выбран Playwright, исходя из следующих характеристик:

- высокая стабильность и минимальное количество нестабильных (flaky) тестов по сравнению с современными аналогами;
- поддержка быстрого параллельного выполнения, что значительно сокращает общее время прогона тестов;
- наличие встроенных механизмов ожидания, уменьшающих количество сбоев, связанных с таймингами и ошибками синхронизации;
- минимальные требования к конфигурации и простая интеграция с Docker и Kubernetes, что соответствует требованиям безопасного запуска автоматизированных тестов в CI/CD-среде;
- наличие мощного API, поддержка React/Vue/Angular, Shadow DOM, управление несколькими окнами, эмуляции и трассировки сетевых запросов, что является обязательным для тестирования на уровне организации.

Выбор инструментов управления тестированием и рабочими процессами. Для управления тестовыми случаями и процессами была выбрана интеграция двух ведущих мировых решений — TestRail и Jira, благодаря

их высокой надёжности, широким интеграционным возможностям и эффективному управлению процессом тестирования.

В качестве DevOps-платформы был выбран GitLab, что обусловлено наличием комплексного набора DevOps-инструментов, объединённых CI/CD-возможностей, управления кодом и автоматизации в рамках одной платформы.

Мониторинг и контроль. Для мониторинга уже развернутой системы используется **Prometheus** — система сбора и хранения метрик. Она предоставляет информацию о загрузке серверов, использовании памяти, показателях CPU и общем состоянии системы. Для визуализации данных применяется инструмент **Grafana**, позволяющий создавать панели мониторинга (dashboards) и систему оповещений для оперативного реагирования на сбои.

Условная формула формирования сигнала оповещения имеет следующий вид:

$$\text{Оповещение} = \begin{cases} 1, & \text{если показатель} \geq \text{порогового значения (threshold)} \\ 0, & \text{если показатель} < \text{порогового значения (threshold)} \end{cases}$$

Полный поток автоматизированной системы. Процесс обеспечения качества в рамках данной системы включает несколько последовательных этапов.

На первом этапе осуществляются уточнение и формализация требований, что позволяет существенно снизить количество потенциальных ошибок на стадии разработки. Данный этап является ключевым в жизненном цикле разработки программного обеспечения (Software Development Life Cycle), поскольку именно от корректности требований зависят качество реализации и оптимизация временных затрат. Неполное или некорректное понимание требований может привести к необходимости повторного прохождения этапов проектирования, разработки и тестирования до формирования окончательной спецификации.

На втором этапе, параллельно с процессом разработки, специалисты по обеспечению качества формируют структуру тестовых наборов, обобщают ожидаемые тестовые случаи и реализуют их автоматизацию. Данный процесс координируется в среде TestRail с чётким соблюдением структуры тестовых наборов и отслеживанием статуса автоматизации каждого тестового случая.

На третьем этапе, после завершения разработки функционала, исходный код передаётся в репозиторий GitLab. Созданный pull request проходит

этапы сборки и деплоя, после чего запускается этап выполнения автоматизированных тестов. Благодаря интеграции GitLab и TestRail автоматически формируется сигнал в TestRail, где создаётся новый тестовый набор для выполнения соответствующих тестовых случаев и регистрации результатов.

Автоматизированное тестирование в рамках pipeline выполняется в headless-режиме, без отображения пользовательского интерфейса браузера. Результаты выполнения каждого тестового случая поэтапно передаются в систему TestRail, где для каждого теста фиксируется статус — успешное выполнение или ошибка. В результате формируется панель результатов, представленная на рис.1.

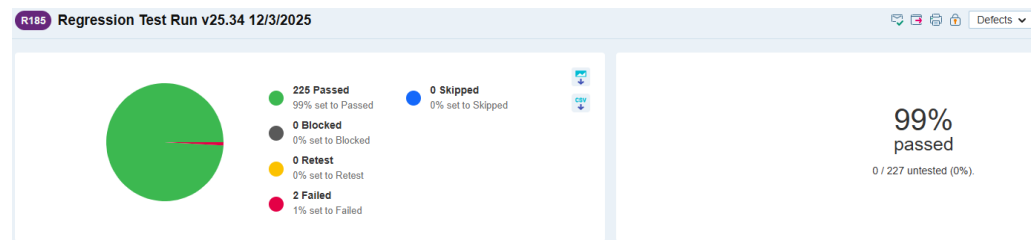


Рис. 1. Панель результатов выполнения тестовых случаев в системе TestRail

Панель результатов (рис.1) сформирована в системе TestRail по итогам выполнения созданных тестовых случаев. Анализ и обобщение полученных результатов позволяют выявить существующие недочёты и ошибки в работе системы.

В среде Playwright, благодаря соответствующим настройкам, в разделе комментариев всех завершившихся с ошибкой тестовых случаев автоматически выводятся данные, зафиксированные в процессе выполнения (логи). Это существенно упрощает процесс выявления причин сбоев и снижает объём рутинной, механически выполняемой работы. На рис.2 представлен список успешных и неуспешных тестовых случаев.

Payment methods 15

ID	Title	Test Labels	Executor	Status	
T205...	Verify payment method functionality by adding Credit Or Debit Card		B	Passed	>
T205...	Verify American Express cards accept 4 digit CVVs		B	Failed	<
T205...	Error messages for invalid Credit Card Numbers		B	Passed	>
T205...	Verify error message displaying in checkout by adding invalid CVV [Updated]		B	Passed	>
T205...	Validate removing Billing Address is not the same as Shipping address checkmark functionality for Credit Or Debit Card		B	Passed	>
T205...	Verify payment method functionality by adding ACH method		B	Passed	>
T205...	Make sure guest users don't see the ACH payment option		B	Passed	>
T205...	Guest user payment method		B	Passed	>
T205...	Validate after removing Save payment method checkmark payment method is not saved		B	Passed	>
T205...	Validate 3D secure card verification is ISuccessful No-Challenge Authentication		B	Failed	>

```

self = <playwright._impl._connection.Connection object at
0x10fb06290>
cb = <function Channel.send.<locals>.<lambda> at
0x10fb2bce0>
is_internal = False

async def wrap_api_call(
    self, cb: Callable[[Any], Any], is_internal: bool = False
) -> Any:
    if self._api_zone.get():
        return await cb()
    task = asyncio.current_task(self._loop)
    at: List[Inspect.FrameInfo] = getattr(task, "_pr_stack_", [])
    inspect_stack()
    parsed_at = _extract_stack_trace_information_from_stack(at, is_internal)
    self._api_zone.set(parsed_at)
    try:
        return await cb()
    except Exception as error:
        raise rewrite_error(error, f"{parsed_at['apiName']}:
{error}") from None
E playwright._impl._errors.TimeoutError:
Locator.wait_for: Timeout 20000ms exceeded
E Call log
E - waiting for locator("tbody>tr:nth(1)") ul >
li") nth(1) to be visible
venv/lib/python3.11/site-
packages/playwright/_impl/_connection.py:528: TimeoutError

```

Рис. 2. Результат неудачного тестового случая и описание причины неудачи в системе TestRail

Как видно на рисунка, при нажатии на неуспешный тест отображается полное описание возникшей ошибки.

Процент успешного выполнения тестового набора можно определить по следующей формуле:

$$S = \frac{T_{\text{успешн}}}{T_{\text{успешн}} + T_{\text{неуспешн}}} \times 100\%,$$

где $T_{\text{успешн}}$ - количество успешно выполненных тестов; $T_{\text{неуспешн}}$ - количество неуспешных тестов; S - процент успешного выполнения всех тестовых случаев набора.

Благодаря интеграции в среде TestRail также реализована возможность быстро и с минимальными действиями отправлять неуспешные тестовые случаи в систему управления задачами Jira и создавать задачу на исправление дефекта (bug), включая в описание тестового случая все шаги и название теста.

Для завершения автоматизированной системы проводится интеграция с Prometheus для сбора метрик и добавляется панель визуализации результатов в Grafana, как показано на рис. 3.

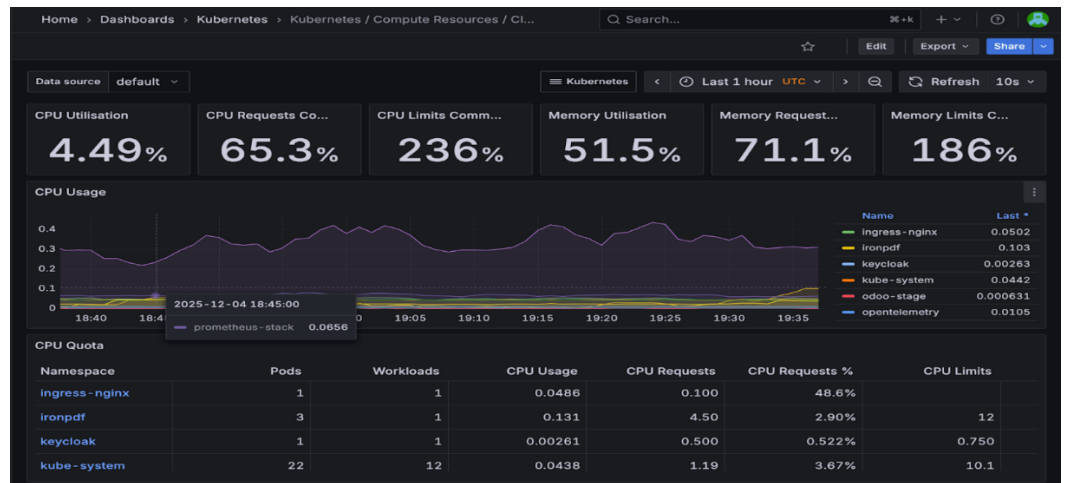


Рис. 3. Графики метрик, сформированные с помощью запросов Prometheus в Grafana

Благодаря панели, представленной на рис.3, осуществляется постоянный мониторинг системы. В случае превышения любого из установленных пороговых значений система оповещений автоматически формирует сигнал о существующей неисправности, который может быть доставлен через электронную почту или опубликован в соответствующих каналах корпоративной коммуникационной платформы.

На рис.4 представлена структура всей автоматизированной системы в соответствии с вышеописанными этапами.

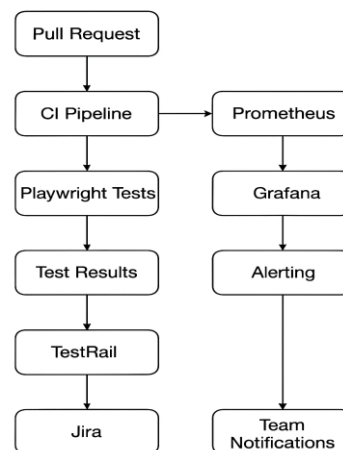


Рис. 4. Полная схема рабочего процесса обеспечения качества

Таким образом, соблюдение последовательности шагов, представленных на рис. 4, позволяет сформировать многофункциональную систему обеспечения качества. В рамках данной системы проверки выполняются на различных этапах разработки, осуществляется мониторинг, создаётся высокоэффективная система, максимально защищённая от критических и повторяющихся ошибок, что обеспечивает стабильный выпуск качественного программного продукта.

Заключение

Обеспечение качества в аппаратно-программных системах требует многослойного подхода, включающего классические методологии QA, автоматизацию, инструменты DevOps, мониторинг и систему оповещений.

Комбинация среды автоматизации Python + Playwright, системы управления тестированием TestRail, Jira, CI/CD-платформы GitLab, а также инструментов мониторинга Grafana/Prometheus позволяет построить полноценную, автономную систему, которая:

- снижает вероятность ошибок, связанных с человеческим фактором;
- обеспечивает быстрое и воспроизводимое тестирование;
- повышает безопасность и качество выпускаемых продуктов;
- улучшает эффективность работы команды.

В результате формируется современная, модульная, надёжная и масштабируемая система обеспечения качества, полностью соответствующая современным инженерным требованиям.

Литература

1. **Kshirasagar Naik, Priyadarshi Tripathy.** Software Testing and Quality Assurance: Theory and Practice.- 2008.- 616 p.
2. **Glenford J. Myers, Corey Sandler, Tom Badgett,** The Art of Software Testing.- 2011.- 256 p.
3. **Cem Kaner, James Bach, Bret Pettichord,** Lessons Learned in Software Testing.- 2002.-320 p.
4. **Vaibhav Devrani,** Automating DevOps with GitLab CI/CD Pipelines.- 2022.- 356 p.
5. **David Burns.** Playwright Testing Handbook.- 2023.- 250 p.
6. **Brian Brazil.** Prometheus: Up & Running: Infrastructure and Application Performance Monitoring.- 2018.- 378 p.
7. **Elton Stoneman.** Grafana 8.x – Real-Time Stream Processing, 2021.- 280 p.
8. **Gurock Software.** TestRail User Guide and Best Practices.- 2023.- 180 p.

*Поступила в редакцию 20.03.2026.
Принята к опубликованию 27.05.2026.*

**ՔՈՄՓՅՈՒԹԵՐԱՅԻՆ ՀԱՄԱԿԱՐԳԵՐԻ ԱՊԱՐԱՏԱԾՐԱԳՐԱՅԻՆ
ՄԻՋՈՑՆԵՐԻ ՈՐԱԿԻ ԱՊԱՀՈՎՄԱՆ ԳՈՐԾԸՆԹԱՑԻ ՀԵՏԱԶՈՏՈՒՄԸ ԵՎ
ԻՐԱԿԱՆԱՑՈՒՄԸ**

Ծ.Ս. Հովհաննիսյան, Ա.Կ. Գալստյան

Ժամանակակից քոմպյուտերային համակարգերի զարգացման ընթացքում ծրագրային ու ապարատային բաղադրիչների որակի ապահովումը (Quality Assurance) հանդիսանում է առանցքային գործընթաց, որը պահպանում է համակարգի հուսալիությունը, ֆունկցիոնալությունը և կայուն աշխատանքը ողջ կենսացիկլի ընթացքում: Ապարատածրագրային լուծումները միավորում են ծրագրային և ֆիզիկական բաղադրիչները, ինչն առաջացնում է ավելի բարդ որակի պահանջներ, քան դասական ծրագրային ապահովման դեպքում:

Ներկայիս արագ զարգացող տեխնոլոգիական աշխարհում ձեռքով կատարվող թեստավորումը այլևս բավարար չէ՝ պայմանավորված մի քանի գործոններով՝ մարդկային սխալներ, մեծ ծավալի թեստային սցենարներ, միջավայրերի բազմազանություն և արագ թողարկումների (Release) պահանջ: Ուստի որակի ապահովման գործընթացներում ավտոմատացումը դարձել է ոչ միայն նպատակահարմար, այլև պարտադիր նախապայման՝ արդյունավետությունն ու ճշգրտությունը բարձրացնելու համար:

Աշխատանքում ներկայացվում են ապարատածրագրային համակարգերի որակի ապահովման հիմնական տեսական սկզբունքները, դրանց կարևորությունը, ինչպես նաև Python + Playwright և մի շարք DevOps գործիքների օգտագործմամբ ավտոմատացված որակի ապահովման համակարգի նախագծումը և իրականացումը:

Առանցքային բառեր. որակի ապահովում, GitLab, DevOps, Prometheus:

**RESEARCH AND IMPLEMENTATION OF A QUALITY
ASSURANCE PROCESS FOR COMPUTER SYSTEM HARDWARE AND
SOFTWARE**

Ts.S. Hovhannisyan, A.K. Galstyan

In the development of modern computer systems, quality assurance of software and hardware components (Quality Assurance) is a key process aimed at guaranteeing the reliability, functionality, and stability of the system throughout its entire lifecycle. Hardware and software solutions integrate software and physical components, leading to more complex quality requirements than traditional software.

With the rapid development of modern technologies, manual testing is no longer sufficient due to a number of factors, including human error, the large volume of test scenarios, the diversity of work environments, and the requirement for rapid release of software products. Therefore, automation of quality assurance processes has become not only advisable but also a prerequisite for improving the efficiency and accuracy of testing.

This paper examines the fundamental theoretical principles of quality assurance for hardware and software systems, their importance, and the design and implementation of an automated quality assurance system using Python, Playwright, and a range of DevOps tools.

Keywords: quality assurance, GitLab, DevOps, Prometheus.