REPORTING & SUMMARY

Наш ключевой подход в этом вопросе - репортинг должен решать ваши проблемы. Если вам достаточно бейджика сборки и логов CI, то не нужно накручивать целую систему сверху.

И уж тем более не нужно "тащить" огромную систему репортинга, если

ваши отчёты никто не использует.

Подключаем Allure

```
plugins {
  id 'java'
  id 'io.qameta.allure' version '2.8.1'
}

...

allure {
  autoconfigure = true
  version = '2.13.0'  // Latest Allure Version

  useJUnit5 {
      version = '2.13.0' // Latest Allure Version
  }
}

repositories {
  jcenter()
  mavenCentral()
}
```

Запуск:
gradlew clean test allureReport
gradlew allureServe
Интеграция с Selendine
dependencies {testImplementation 'io.qameta.allure:allure-selenide:2.13.0'}

В тестах:

```
@BeforeAll
static void setUpAll() {
  SelenideLogger.addListener("allure", new AllureSelenide());
}

@AfterAll
static void tearDownAll() {
  SelenideLogger.removeListener("allure");
}
```

Allure позволяет гораздо больше, чем "просто логгировать" - на самом деле,

вы можете отмечать нужные события прямо в тестах, прикладывать скриншоты, файлы и помечать тесты.

Обратитесь к документации за более подробной информацией.

Автоматизация не самоцель.

Автоматизируя что-то, вы должны чётко понимать какие проблемы тем самым решаете.

Если у вас нет проблем, то, возможно, и автоматизировать ничего не стоит.
Цели автоматизации могут включать в себя следующие:

— Сокращение стоимости тестирования — замещение некоторых ручных операций делает процесс дешевле;

— Сокращение времени на тестирование — можем тестировать быстрее;

— Увеличение покрытия тестирования — можем тестировать больше функциональности;

— Проведение особых видов тестирования, которые человек не в состоянии провести — например,
нагрузочное и т.д.;

— Увеличение частоты тестирования — особенно важно в гибких методологиях с несколькими релизами в
день;

— Быстрая обратная связь

— Исключение человеческого фактора при ручном тестировании (утомляемость,

невнимательность и т.д.)
Автоматизация заключается не только в автоматическом прогоне тестов.

Вы можете автоматизировать подготовку тестового окружения (привет Docker!), тестовых данных,
репортинг и любую активность в рамках

процесса тестирования.

Автоматизация требует дополнительных навыков и квалификации.

Вы должны:

— уметь программировать

— разбираться в экосистеме языка (платформе)

— иметь опыт работы с ключевыми библиотеками и инструментами

— постоянно следить за развитием (языка, инструментов и всего остального).

A Test Automation Engineer is one who has broad knowledge of testing in general, and an in-depth understanding in the specific area of test automation.

Вопрос "что автоматизировать в первую очередь" на самом деле не такой уж тривиальный.

Регресс или новые фичи?

Мы можем автоматизировать "регресс"? Но ведь там уже всё может быть достаточно спокойно (ключевые
баги выловлены) и автоматизируя регресс вы всегда будете на шаг позади разработки.

Мы можем автоматизировать "новые фичи"? Но ведь они не стабильны, и нам достаточно часто придётся
переписывать тесты.

Идеального решения не существует - в каждом проекте решение своё.

ПРОБЛЕМЫ АВТОМАТИЗАЦИИ

1. Много тестов - тяжело поддерживать, модифицировать

2. Мало тестов - покрывают малую часть функциональности

3. Медленные тесты - никто не запускает, т.к. "слишком медленно", нет быстрой обратной связи

4. Устаревшие тесты - ложные срабатывания, отстутствие доверия к системе (не успеваем следить за
изменением продукта и выкидывать не нужное)

Тесты - это тоже код. Не важно, это код авто-тестов или конфигурация,
которая позволяет вам запустить Docker Compose. Код нуждается в поддержке и рефакторинге.

Точно так же накапливается технический долг, который требует устранения
(либо вы получите "неподдерживаемое нечто").

Отдельная большая проблема - насколько вам легко получать (или генерировать тестовые данные).
А что, если ваша система проверяет номера паспортов или ИНН?
Насколько легко вам будет получить валидные данные (особенно если есть интеграция с гос.органами)?
И позволяет ли ваша система "подставить заглушки" для сгенерированных данных.
Ключевая проблема - позволяет ли ваша система быстро и, самое главное,приводить её в нужное состояние?
Уверены ли вы, что приведя её в нужное состояние (например, через прямой доступ к СУБД), вы тестируете реальное поведение системы, а не"сломанную" неконсистентностью логику?

УРОВНИ ТЕСТИРОВАНИЯ

— Manual — ручные тесты, их должно быть меньше всего (т.к. плохо масштабируются);

— GUI — тестирование через графический интерфейс;

— API — тестирование через API (например, REST API);

— Unit — изолированное тестирование отдельно взятого программного компонента.

Инструменты каждого уровня вносят собственную "вероятность ошибки"- делая систему тестирования более хрупкой. Это нужно учитывать, распределяя усилия.
Но кроме того, нужно помнить, что Unit тесты или тесты API не дают вам понимания того, что система работает "в целом".
Они дают лишь фундамент для того, чтобы писать меньше тестов верхнего уровня (и меньше тестировать руками).

Вы должны понимать, что самые критичные сценарии должны быть покрыты E2E-тестами, имитирующими поведение реального пользователя.
Ключевое - именно реального. Мы почти на каждом проекте сталкиваемся с тем, что сценарий автоматизируется исходя из того, как автоматизатор представляет поведение пользователя.
Запомните: если вы не знаете, как ведёт себя реальный пользователь (у вас нет данных аналитики), то вы автоматизируете собственные догадки!

И пожалуйста, старайтесь избежать синдрома эксперта*, когда вы и без реального пользователя "всё знаете" о пользователе и его поведении - потом очень больно будет признавать, что вы со своим "всё знаю" были вкорне не правы.

Автоматизация не ограничивается только функциональным тестированием, поэтому обязательно расширяйте свой кругозор на предмет других видов:
— производительности

— безопасности— и т.д.

Инструменты - ключевая часть автоматизации. Правильно подобранные и удобные - ключ к успеху.

Поэтому внимательно изучайте существующие и постоянно отслеживайте появление новых.

Docker, TestContainers, Selenide, Selenoid, JUnit, Gradle - это лишь ваш базовый набор, за которым
вы должны следить (а обновления в современном мире выходят чуть ли не раз в месяц).

Кроме того, обязательно практикуйтесь в том, что не устаревает: Linux,

Windows, Bash, PowerShell, CMD, Git.

ИНТЕГРАЦИИ - Одна из самых сложных частей, т.к. почти не осталось самодостаточных систем.
Большая часть так или иначе интегрируется с другими системами для выполнения необходимых функций.
Вы должны всегда выяснять, с какими системами интегрирована ваша, и как вы можете "вклиниться" в
эту схему для проведения тестирования:
— изолированного (только вашей системы)
— интеграционного (inter-system communication)

Отчётность
и её историческая хронология должна быть, чтобы вы
понимали тренды.
Не важно, как это реализовано (логи сборки в CI), Allure с интеграцией с CI или Report Portal.
Ключевое правило тестирования: если вы ничего не измеряете, не храните историю, и по результатам
периодического анализа не вносите коррективы, то вы не выстраиваете процесс.
Нет процесса - нет результата.