Маніфест QA Automation (SDET)

як рушій безперервного підвищення якості

Роман Поботін

Вступ

Сучасна розробка ПЗ вимагає від тестувальників значно ширшого набору навичок, ніж просто знання базових методів тестування. Роль **QA Automation (SDET)** — це цілісне об'єднання декількох напрямів:

- **QA** (розуміння видів тестування, підходів і процесів),
- **Development** (глибокі навички програмування, патерни, архітектура),
- **DevOps** (CI/CD, контейнеризація, моніторинг, логування).

Основна ідея цього Маніфесту: **QA Automation — не лише «перевірка продукту», а й оптимізація всього циклу розробки для досягнення швидких і якісних релізів**, із урахуванням підходів SDET, керування тестами (ISTQB® CTAL-TM) та побудови структурованої автоматизації (ISTQB® CTAL-TAE). Також у цьому документі розглядаються ідеї **Behavior-Driven Development** (Dan North), **Test Automation Manifesto** (Smith & Meszaros) та головні тези з **xUnit Test Patterns**.

1. Основні принципи

1.1. QA Automation — не просто «тести», а оптимізація процесів

1. **Цілісне бачення розробки** Автоматизатор має розуміти весь цикл: від ідеї та дизайну до деплойменту й підтримки. Сприяє виявленню «вузьких місць», покращує командну взаємодію, пришвидшує *delivery*.
2. **Прискорення та гнучкість** Правильно налаштовані CI/CD-пайплайни, DevOps-інструменти та автоматизація рутинних операцій дають змогу швидко випускати оновлення без втрати якості.
3. **Зменшення ручної роботи** Мета — позбутися «клікаторства» та повторюваних завдань. Все, що можна автоматизувати, має бути автоматизовано.

1.2. SDET: перетин QA, Development, DevOps

- **QA-аспект:**

  - Знання різних видів тестів: юніт, інтеграційних, E2E, перформанс, безпека.
  - Уміння розробляти тестову стратегію, пропонувати покращення процесів (TestOps).

- **Development-аспект:**
  - Володіння мовами програмування (бажано декількома), знання алгоритмів, структур даних, ООП, SOLID, KISS, DRY, YAGNI тощо.
  - Розуміння патернів проєктування (Page Object, Factory, Singleton), уміння створювати додаткові інструменти (генератори даних, кастомні репортери, логгери).

- **DevOps-аспект:**
  - Налаштування CI/CD (Jenkins, GitLab CI, GitHub Actions, Azure DevOps).
  - Контейнеризація (Docker, Kubernetes), інтеграція з моніторингом, логуванням.
  - Вміння оперативно доставляти продукт і відслідковувати його стабільність.

1.3. Програмування як обов'язкова складова

1. **Тести + інструменти** Автоматизатор не повинен обмежуватися написанням тестів. Він має створювати скрипти, утиліти та фреймворки, що допомагають у тестуванні та розвитку проєкту загалом.

2. **Робота з різними мовами** Швидка адаптація до нових технологій: React, Angular, Python, Go, Java — залежно від потреб проєкту.

3. **Постійна практика** Рішення алгоритмічних задач (LeetCode, CodeWars) підтримує форму та дозволяє швидко писати якісний код навіть у стресових ситуаціях.

1.4. T-Shape, TestOps та аналітика

1. **T-Shape фахівець** «Вертикаль» — глибокі знання тестування (фреймворки, методології). «Горизонталь» — базове чи середнє розуміння суміжних сфер: архітектура, DevOps, UX, бази даних.

2. **TestOps та моніторинг** Налаштування метрик: coverage, build health, release frequency, defect leakage, MTTR. Аналітика в реальному часі: відстеження продуктивності та стабільності тестового середовища. Постійне покращення процесів, а не латання точкових «дір».

3. **Безперервне вдосконалення** Культура регулярного покращення якості: аналіз, ретроспективи, швидкі експерименти та впровадження корисних змін.

2. BDD: Правильне використання або взагалі не використання

Загальна ідея BDD

BDD (*Behavior-Driven Development*) — це методологія, що спирається на тісну комунікацію між бізнес-сторонами (Product Owner, Business Analyst), розробниками й тестувальниками. Згідно з *Introducing BDD (Dan North)*, основна увага приділяється поведінці системи, а не реалізації.

- **Спільна мова (Ubiquitous Language):** Учасники працюють над *Given / When / Then* прикладами, що описують *що* система має робити, а не *як*.
- **Фокус на «прикладах»**: Приклади (Scenario) описуються мовою бізнесу, уникаючи технічних деталей.
- **Спільна відповідальність**: Усі зацікавлені сторони формулюють правила поведінки та приклади. QA орієнтується на підтвердження їх виконання, а розробники — на реалізацію.

Правильне використання BDD

- **Сценарії на випередження:** Сценарії зазвичай пише PO або BA, а розробники й тестувальники уточнюють деталі.
- **Впровадження функціоналу за попередніми сценаріями:** Якщо тест (Given / When / Then) успішний — продукт відповідає очікуванням.
- **Уникайте «штучного» BDD:** Якщо команда не має можливості (чи бажання) реалізувати справжній цикл співпраці, краще обійтися більш простим форматом тестів.

3. Test Cases as Code

- Тест-кейси мають бути *інтегровані* в код, щоб уникнути дублювання інформації і «синхронізації» між тест-менеджмент-системами та автотестами.
- Оновлення тестів відбувається одночасно з оновленням коду: це знижує ризик розсинхронізації та сприяє прозорості.

4. Культура співпраці

1. **QA як частина команди розробки**

- QA — це не стороння роль, а ключовий учасник команди, який формує якість продукту.
- Тестувальник — не той, хто «чекає на функціонал, щоб гавкати на помилки». Це *інженер*, що допомагає будувати рішення, передбачає і усуває ризики **на ранніх етапах**.

2. **TDD як ціль**
   - QA сприяє впровадженню TDD (Test-Driven Development), коли розробники пишуть *юніт*-тести ще до реалізації функціоналу.
   - Знижує кількість дефектів «у полі» і робить продукт більш передбачуваним у розробці.

5. Утопічна ідея: «QA, що виконав місію, може йти далі»

1. **Навчити команду «жити без QA»** Ідеальний сценарій: розробники розуміють цінність тестів і пишуть їх самі, а процеси настільки відпрацьовані, що немає потреби в постійному ручному контролі.
2. **Спільна відповідальність** «За якість відповідає лише QA» — шкідливе мислення. Мета — *shared responsibility*, де кожен дбає про якість.
3. **Роль консультанта** QA стає «позаштатним» ментором, до якого звертаються, коли потрібно розширити тестову стратегію чи побудувати нову систему моніторингу.

6. Team Vision: shift-left та глибша інтеграція

1. **Engineer First, Not Just Testers** Усі QA-інженери розглядаються як повноцінні інженери. Менше ручних перевірок, більше технічних навичок та повноцінного розвитку.
2. **T-Shape Skill Development** Кожен член команди є «універсальним солдатом» з глибокою основною спеціалізацією (наприклад, у тестуванні) та широким спектром знань у суміжних областях (DevOps, Data, UI тощо).
3. **Everything is Possible to Automate** Якщо зустрічаємо задачу, яку на перший погляд неможливо автоматизувати, треба шукати творчі рішення або чітко доводити неможливість.
4. **Adaptability to New Technologies** Використання ШІ (наприклад, ChatGPT) для прискорення пошуку рішень, експериментів із новими бібліотеками та фреймворками.
5. **Enthusiasm and Ownership** Ніхто не каже «Це не моя робота». Якщо проблема стосується загального процесу, кожен має право та можливість її вирішувати.

7. Приклади з реальних ситуацій (без коду)

1. Оптимізація регресійних тестів

Ситуація: Тривалий регрес призводить до затримки релізів. **Дії QA/SDET:** Налаштування паралельного запуску та поділ тестів на *smoke/регрес*. Smoke-тести запускаються на кожен коміт, а повний регрес — після закриття великих задач. **Результат:** Скорочення часу зворотного зв'язку, швидша доставка функціоналу.

2. Загальна відповідальність за якість

Ситуація: У команді вважають, що «тести — це зона QA». Результат — пропускання дефектів, які могли б виловити розробники на рівні юніт-тестів. **Дії:** QA проводить воркшопи з TDD, навчає розробників писати базові тести. **Результат:** Менше дефектів «просочується» далі, а QA може сконцентруватися на складніших інтеграційних сценаріях.

3. Покращення build health

Ситуація: Часті «червоні» збірки через нестабільні (flaky) тести. **Дії:** Запровадження маркерів для flaky-тестів, регулярний аналіз логів, виправлення або видалення ненадійних сценаріїв. **Результат:** Стабільні пайплайни, менше фальшивих «аварій», швидший цикл зворотного зв'язку.

8. Додаткові ідеї з ISTQB Advanced Level Test Automation Engineer (TAE)

Нижче викладено кілька ключових моментів із ISTQB CTAL-TAE:

1. **Business Case for Automation** Автоматизація має бути *економічно виправданою*.
   - Визначення потенційного скорочення витрат.
   - Можливі вигоди (прискорення релізів, зниження ручної рутини).
2. **Automation Architecture (TAA)**
   - Використовувати модульний підхід (**Layered Architecture**: шари для управління даними, логікою тестів).
   - *Design for Testability*: продукт має бути тестопридатним (mock-и, API-хуки).
3. **Implementation and Integration**
   - **Вибір фреймворку:** мова, можливості звітності, CI/CD.
   - **Pipeline:** безперервна інтеграція, регулярні запуски.

4. **Metrics and Monitoring**
    - *Coverage*, *defect detection rate*, *flake rate*.
    - *MTTA* (Mean Time To Acknowledge), *MTTR* (Mean Time To Repair).
5. **Maintaining the Automated Solution**
    - Регулярний **рефакторинг** тестів, оновлення селекторів.
    - Відстежування версій тестів разом із версіями продукту.
6. **Transition Manual to Automated**
    - Не варто *слпо* конвертувати всі мануальні кейси.
    - Почати з регресійних тестів, розширювати автоматизацію поступово.

9. Ідеї з ISTQB® Certified Tester Advanced Level Test Manager (v3.0)

Нижче викладено тези з ISTQB CTAL-TM, що доповнюють наш Маніфест принципами управління тестуванням:

1. Призначення автоматизації тестування

- **Повторюваність та консистентність:** Тести на різних версіях ПЗ і середовищах повинні виконуватися передбачувано.
- **Мета автоматизації:**
    - Підвищення ефективності тестування.
    - Розширення охоплення функціональності.
    - Зниження загальної вартості тестування.
    - Виконання тестів, які неможливо реалізувати вручну (паралельні, реального часу).
    - Скорочення часу на тестування, збільшення частоти тестувань.

2. Фактори успіху автоматизації

1. **Архітектура автоматизації тестування (TAA):**
    - Проєктування з урахуванням вимог до підтримки, продуктивності, простоти навчання.
    - Забезпечення сумісності між TAS (Test Automation Solution) та архітектурою SUT.

2. **Тестованість SUT:**
    - Розділення GUI від бізнес-логіки.
    - Публічність API та інтерфейсів для тестування.
3. **Стратегія автоматизації тестування:**
    - Практичний підхід, що враховує підтримку, витрати й ризики.
    - Комбінація GUI та API-тестів для перевірки консистентності.
4. **Фреймворк автоматизації (TAF):**
    - Легкість використання, документованість, модульність.
    - Підтримка звітності та налагодження.
5. **Підтримка середовища тестування:**
    - Контрольоване середовище, моніторинг і відновлення SUT під час тестів.

3. Ризики та обмеження автоматизації

- Висока початкова вартість розробки TAS.
- Не всі ручні тести можна автоматизувати.
- Залежність від змін у середовищі/інтерфейсах SUT.
- Не замінює дослідницьке/евристичне тестування.

4. Ефективна архітектура TAS

1. **Багаторівнева архітектура TAS:**
    - Шар генерації тестів: дизайн кейсів, авто-генерація на основі моделей.
    - Шар визначення тестів: процедури та дані для тестів.
    - Шар виконання тестів: запуск, логування результатів.
    - Шар адаптації: інтеграція з SUT (API, протоколи, емулятори).
2. **Принципи архітектури:**
    - Single Responsibility.
    - Розширюваність без модифікацій.
    - Абстракція для сумісності з різними інструментами.

5. Метрики та звітність

- Відстеження продуктивності TAS, часу виконання, покриття, логування.
- Звіти: надавати стейкхолдерам інформативні дані про стан якості.

6. Перехід від ручного до автоматизованого тестування

- Визначення критеріїв для автоматизації (стабільність, обсяг, частота).
- Почати з регресійних тестів, далі — нові функції.
- Використання негативних тестів.

7. Безперервне вдосконалення TAS

- Постійне оновлення тестів відповідно до змін у SUT.
- Видалення застарілих тестів, оптимізація процесів.

8. Взаємодія автоматизації з SUT

- Використання різних рівнів доступу: GUI, API, протоколи, сервіси.
- Дизайн SUT для тестованості (додаткові інтерфейси, моніторинг станів).

9. Ключові підходи до автоматизації

- Capture/Playback (початкові записи взаємодій).
- Data-Driven Testing (дані в окремих файлах).
- Keyword-Driven Testing (використання ключових слів у сценаріях).

10. Test Automation Manifesto (2003, Smith & Meszaros)

Шон Сміт та Жерар Месзарош представили «Маніфест автоматизації тестування» на конференції XP/Agile Universe у 2003 році. Основні принципи:

1. **Писати тести спочатку (Write the Tests First)** Розробка тестів перед написанням коду сприяє створенню тестованого дизайну та зменшує витрати на налагодження.
2. **Проектувати для тестованості (Design for Testability)** Забезпечення того, щоб система була легко тестованою, спрощує процес автоматизації та підвищує якість тестів.

3. **Використовувати основний інтерфейс (Use the Front Door First)** Тестування через публічний інтерфейс системи знижує залежність від внутрішньої реалізації й підвищує надійність тестів.

4. **Передавати намір (Communicate Intent)** Тести мають бути зрозумілими та відображати очікувану поведінку, що полегшує їх підтримку й використання як документації.

5. **Не модифікувати тестовану систему (Don't Modify the SUT)** Уникати змін у системі під час тестування, щоб упевнитися, що перевіряється реальна поведінка продукту, а не його кастомізована версія.

6. **Тримати тести незалежними (Keep Tests Independent)** Кожен тест мусить бути автономним, щоб зміни в одному тесті не впливали на інші — це забезпечує стабільність та надійність набору.

Дотримання цих принципів сприяє створенню ефективної та стійкої системи автоматизованого тестування, підтримує високу якість ПЗ та спрощує розробку.

11. Додаткові ідеї з "xUnit Test Patterns" (G. Meszaros)

Chapter 3: Goals of Test Automation

- **Швидкий зворотний зв'язок (Fast Feedback):** Тести мають виконуватися швидко, даючи миттєву оцінку коду.
- **Попередження дефектів (Defect Prevention):** Регулярний запуск тестів допомагає запобігати інтеграційним проблемам.
- **Покращення якості коду (Improved Code Quality):** Наявність автотестів стимулює розробників писати більш чистий і модульний код.
- **Економія ресурсів (Cost Reduction):** Хоча стартові витрати можуть бути високими, у довгостроковій перспективі автоматизація знижує загальні витрати.
- **Повторюваність і надійність (Repeatability and Reliability):** Автотести мають однакові результати при кожному запуску.
- **Можливість рефакторингу (Facilitate Refactoring):** З наявними тестами розробники впевненіше змінюють код.
- **Підтримка Agile-практик:** Часті релізи, continuous integration тощо.
- **Документація поведінки (Documenting Behavior):** Тести описують очікувану поведінку системи в реальному часі.

Chapter 4: Philosophy of Test Automation

- **Тести як актив (Tests as Assets):** Вартує розглядати тести як довгострокову інвестицію.
- **Стійкість до змін (Resilience to Change):** Тести мають бути гнучкими до змін у коді.
- **Постійна цінність (Sustained Value):** Навіть через роки автотести лишаються корисними.
- **Уникнення пасток (Avoiding Automation Pitfalls):** Не всі проблеми вирішує автоматизація, слід уникати дублювання й надмірної складності.
- **Пріоритет стабільності (Prioritize Stability):** Ніяких фальшивих позитивів чи негативів — тести мають бути надійними.
- **Мінімізація вартості володіння (Minimizing Cost of Ownership):** Час на написання, запуск і підтримку тестів повинен окупатися.
- **Спільна відповідальність (Shared Responsibility):** Уся команда має підтримувати тести.

Chapter 5: Principles of Test Automation

- **Окремий тестовий код (Separate Test Code):** Тестовий код відокремлений від основного.
- **Незалежність тестів (Test Independence):** Тести не повинні залежати один від одного.
- **Повторюваність (Repeatability):** Результати мають бути однаковими за будь-яких умов.
- **Швидкий зворотний зв'язок (Fast Feedback):** Чим швидше виконується, тим корисніше.
- **Простота і зрозумілість (Simplicity and Clarity):** Тести часто слугують «живою документацією».
- **Захищеність від змін (Resistance to Change):** Мінімізувати оновлення тестів після кожної зміни коду.
- **Сфокусованість на бізнес-цінності (Focus on Business Value):** Перевіряти насамперед критичний для бізнесу функціонал.
- **Обробка аномалій (Robustness):** Тести мають коректно реагувати на несподівані стани.

- **Модульність і повторне використання (Modularity and Reusability):** Поділ тестів на логічні компоненти.
- **Інструменти автоматизації (Tool Selection):** Обирати інструменти під конкретні потреби.

12. Корисні ресурси

Книжки

- *Full Stack Testing: A Practical Guide for Delivering High Quality Software* — Amazon
- *xUnit Test Patterns: Refactoring Test Code* — Amazon
- *Clean Agile: Back to Basics* — Amazon
- *Accelerate: The Science of Lean Software and DevOps* — Amazon

Онлайн-спільноти та платформи

- **Ministry of Testing** — ministryoftesting.com
- **SQA StackExchange** — sqa.stackexchange.com
- **GitHub Actions Marketplace** — github.com/marketplace?type=actions

Для прокачування технічних навичок

- **LeetCode** — leetcode.com
- **Codewars** — codewars.com
- **Real Python** — realpython.com

13. Висновок

QA Automation (SDET) — це не «людина з чеклістом» чи «клікер», а рушійна сила безперервного поліпшення якості. Цей фахівець:

- Поєднує тестування, розробку та DevOps-практики.
- Забезпечує прозорість процесів і реальну аналітику (метрики, моніторинг, логування).
- Робить внесок у стабільність та швидкість релізів (через налаштовані CI/CD, автоматизацію рутин).

- Навчає команду брати відповідальність за якість і створювати перевіряльний код від самого початку.

Ідеальний результат роботи SDET — коли команда настільки *виросла* в культурі якості, що може самостійно підтримувати високий рівень тестування і розробки навіть без постійної присутності QA-фахівця. Застосування принципів **ISTQB CTAL-TAE** (ефективна архітектура TAS, TestOps, безперервне покращення) та **ISTQB CTAL-TM** (ризик-орієнтоване тестування, стратегія, залучення стейкхолдерів), концепцій **BDD** (Dan North), ідей **Test Automation Manifesto** (Smith & Meszaros) та **xUnit Test Patterns** створює стійкий фундамент для **безперервного розвитку** й **високої якості** програмного забезпечення.

Цей документ об'єднує:
принципи SDET, BDD, ISTQB Advanced TAE, CTAL-TM, Test Automation Manifesto, тези з xUnit Test Patterns, приклади з реальних проєктів та джерела для глибшого вивчення.
2025