surf.ru

Как мы автоматизировали тестирование аналитики в мобильном приложении

Сергей Лазарев
QA Surf

Проблема

В процессе разработки своих приложений мы столкнулись с тем, что тестирование аналитики требует большого количества ресурсов.

Кроме того, ручное тестирование не всегда может обеспечить тот уровень покрытия, который мы считаем необходимым.

Предложенное решение проблемы не ориентируется на конкретную систему аналитики, платформу и стек автоматизации и может быть реализовано для большинства приложений.

О чём сегодня поговорим

1. Что такое событие аналитики и какова его роль в приложении
2. Как происходит ручное тестирование аналитики
3. Автоматизация тестирования отдельного события
4. Алгоритм полного тестирования всех событий в приложении
5. Преимущества и недостатки описанного подхода
6. Анализ затрат времени
7. Сравнение ручного и автоматизированное тестирования аналитики
8. Выводы

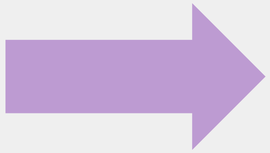
Что такое событие аналитики и какова его роль в приложении

Событие аналитики дает информацию о совершении пользователем в приложении определенного действия, а также о параметрах этого действия.

Совокупность отправленных событий позволяет проанализировать опыт пользователя и повысить эффективность приложения с точки зрения бизнеса.

Что такое событие аналитики и какова его роль в приложении

Пользователь добавляет товар в корзину



“В корзину добавлен товар с id {product_id}”

Что такое событие аналитики и какова его роль в приложении

Пользователь добавляет товар в корзину	“В корзину добавлен товар с id {product_id}”
Пользователь оформляет заказ	“Оформлен заказ с товаром id {product_id}”

Как происходит ручное тестирование аналитики

Создание набора тест кейсов

Как происходит ручное тестирование аналитики

Создание набора тест кейсов

Вызов событий в рамках теста

Как происходит ручное тестирование аналитики

Создание набора тест кейсов

Вызов событий в рамках теста

Верификация каждого события

Автоматизация тестирования отдельного события

Отдельные условия:

1. В приложении реализованы “сценарные” e2e-автотесты (это важно для переиспользования их шагов и сокращения количества кода, необходимого для тестирования непосредственно аналитики)

2. Технический стек проекта позволяет из теста получить доступ к отправляемым событиям и их параметрам

3. Наличие возможности безопасно запускать такие тесты в каком-либо окружении, чтобы не портить реальные данные.

Автоматизация тестирования отдельного события

Рассматриваемый стек:

1. мобильное приложение на Flutter
2. нативные e2e-тесты
   - Gherkin для структуры + Cucumber для отчетов
   - библиотеки integration_test, surf_flutter_test, mocktail, equatable

Автоматизация тестирования отдельного события

Пример события в продуктовом коде:

```
factory AnalyticsEvent.orderCardOpened({
    required String orderId,
    required String store,
  }) =>
      _OrderCardOpened(
        orderId: orderId,
        store: store,
      );
```

Локальная проверка отдельного события

```
final analyticsStepDefinitions = [
    testerThen<ContextualWorld>(
      RegExp(r'I succeed with sending analytic event orderCardOpened$'),
        (context, tester) async {
       await tester.pumpUntilVisible(ordersListTestScreen.trait);
       await tester.implicitTap(firstOrderCard);
       await tester.pumpUntilVisible(orderCardTestScreen.trait);
       final analytics = context.world.container.read(mockAnalyticsProvider);
       verify(() => analytics.logEvent(AnalyticsEvent.orderCardOpened(
          orderId: '12345678',
          store: 'Test store',
        ))).called(1);
      },
    ),
  ];
```

Локальная проверка отдельного события

```
final analyticsStepDefinitions = [
    testerThen<ContextualWorld>(
      RegExp(r'I succeed with sending analytic event orderCardOpened$'),
        (context, tester) async {
       await tester.pumpUntilVisible(ordersListTestScreen.trait);
       await tester.implicitTap(firstOrderCard);
       await tester.pumpUntilVisible(orderCardTestScreen.trait);
       final analytics = context.world.container.read(mockAnalyticsProvider);
       verify(() => analytics.logEvent(AnalyticsEvent.orderCardOpened(
          orderId: '12345678',
          store: 'Test store',
        ))).called(1);
      },
    ),
  ];
```

Локальная проверка отдельного события

```
final analyticsStepDefinitions = [
    testerThen<ContextualWorld>(
      RegExp(r'I succeed with sending analytic event orderCardOpened$'),
        (context, tester) async {
       await tester.pumpUntilVisible(ordersListTestScreen.trait);
       await tester.implicitTap(firstOrderCard);
       await tester.pumpUntilVisible(orderCardTestScreen.trait);
       final analytics = context.world.container.read(mockAnalyticsProvider);
       verify(() => analytics.logEvent(AnalyticsEvent.orderCardOpened(
          orderId: '12345678',
          store: 'Test store',
        ))).called(1);
      },
    ),
  ];
```

Локальная проверка отдельного события

```
final analyticsStepDefinitions = [
    testerThen<ContextualWorld>(
      RegExp(r'I succeed with sending analytic event orderCardOpened$'),
        (context, tester) async {
       await tester.pumpUntilVisible(ordersListTestScreen.trait);
       await tester.implicitTap(firstOrderCard);
       await tester.pumpUntilVisible(orderCardTestScreen.trait);
       final analytics = context.world.container.read(mockAnalyticsProvider);
       verify(() => analytics.logEvent(AnalyticsEvent.orderCardOpened(
          orderId: '12345678',
          store: 'Test store',
        ))).called(1);
      },
    ),
  ];
```

Автоматизация тестирования отдельного события

Переиспользование шагов из e2e-тестов позволяет дополнительно создавать только целевые шаги с непосредственным вызовом и проверкой события.

Для реализации такого переиспользования иногда требуется в этих шагах после вызова `verify()` выполнять дополнительные действия, не относящиеся к проверке события.

Автоматизация тестирования отдельного события

Недостатки такого подхода:

1. целевое событие может быть вызвано до целевого действия
2. целевое событие может быть отправлено второй раз,
   но с неверными параметрами
3. целевое действие может вызвать отправку
   не предусмотренного события
4. в ходе сценария могут быть отправлены непредусмотренные
   события

Алгоритм полного тестирования всех событий в приложении

1. Создаем общую функцию `checkAnalyticsEvent()`, которая может обрабатывать любое событие.
2. В тестовом шаге кладем все необходимые параметры в переменные.
3. Заменяем вызов `verify()` на вызов `checkAnalyticsEvent()`, передавая заданные ранее переменные в качестве параметров.

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Общая функция для проверки событий

```
Future<void> checkAnalyticsEvent(
  MockAnalyticsService analytics,
  Future<void> Function() actionToCallEvent,
  AnalyticsEvent event,
  int eventsCalled,
) async {
  verifyNever(() => analytics.logEvent(event));
  if (eventsCalled == 0) {
    verifyNever(() => analytics.logEvent(any()));
  } else {
    verify(() => analytics.logEvent(any())).called(eventsCalled);
  }
  await actionToCallEvent();
  verify(() => analytics.logEvent(event)).called(1);
  verifyNever(() => analytics.logEvent(any()));
}
```

Алгоритм полного тестирования всех событий в приложении

1. Создаем общую функцию `checkAnalyticsEvent()`, которая может обрабатывать любое событие.
2. В тестовом шаге при необходимости кладем все нужные параметры в переменные.
3. Заменяем вызов `verify()` на вызов `checkAnalyticsEvent()`, передавая заданные ранее переменные в качестве параметров.

Вызов функции в шаге теста

```
await tester.implicitTap(ordersTestScreen.firstOrderCard);
await tester.pumpUntilVisible(orderCardTestScreen.trait);
final AnalyticsEvent event = OrdersAnalyticsEvent.orderCancelClicked(
  orderId: '12345678',
);
await tester.checkAnalyticsEvent(
  context.world.container.read(mockAnalyticsProvider),
  () async {
    await tester.implicitTap(orderCardTestScreen.orderSecondaryButton);
    await tester.pumpUntilVisible(generalTestScreen.actionBottomSheet);
  },
  event,
  // Should be events LoggedIn, OrdersOpened, OrderCardOpened.
  3,
);
```

Вызов функции в шаге теста

```
await tester.implicitTap(ordersTestScreen.firstOrderCard);
await tester.pumpUntilVisible(orderCardTestScreen.trait);
final AnalyticsEvent event = OrdersAnalyticsEvent.orderCancelClicked(
  orderId: '12345678',
);
await tester.checkAnalyticsEvent(
  context.world.container.read(mockAnalyticsProvider),
  () async {
    await tester.implicitTap(orderCardTestScreen.orderSecondaryButton);
    await tester.pumpUntilVisible(generalTestScreen.actionBottomSheet);
  },
  event,
  // Should be events LoggedIn, OrdersOpened, OrderCardOpened.
  3,
);
```

Вызов функции в шаге теста

```
await tester.implicitTap(ordersTestScreen.firstOrderCard);
await tester.pumpUntilVisible(orderCardTestScreen.trait);
final AnalyticsEvent event = OrdersAnalyticsEvent.orderCancelClicked(
  orderId: '12345678',
);
await tester.checkAnalyticsEvent(
  context.world.container.read(mockAnalyticsProvider),
  () async {
    await tester.implicitTap(orderCardTestScreen.orderSecondaryButton);
    await tester.pumpUntilVisible(generalTestScreen.actionBottomSheet);
  },
  event,
  // Should be events LoggedIn, OrdersOpened, OrderCardOpened.
  3,
);
```

Вызов функции в шаге теста

```
await tester.implicitTap(ordersTestScreen.firstOrderCard);
await tester.pumpUntilVisible(orderCardTestScreen.trait);
final AnalyticsEvent event = OrdersAnalyticsEvent.orderCancelClicked(
  orderId: '12345678',
);
await tester.checkAnalyticsEvent(
  context.world.container.read(mockAnalyticsProvider),
  () async {
    await tester.implicitTap(orderCardTestScreen.orderSecondaryButton);
    await tester.pumpUntilVisible(generalTestScreen.actionBottomSheet);
  },
  event,
  // Should be events LoggedIn, OrdersOpened, OrderCardOpened.
  3,
);
```

Вызов функции в шаге теста

```
await tester.implicitTap(ordersTestScreen.firstOrderCard);
await tester.pumpUntilVisible(orderCardTestScreen.trait);
final AnalyticsEvent event = OrdersAnalyticsEvent.orderCancelClicked(
  orderId: '12345678',
);
await tester.checkAnalyticsEvent(
  context.world.container.read(mockAnalyticsProvider),
  () async {
    await tester.implicitTap(orderCardTestScreen.orderSecondaryButton);
    await tester.pumpUntilVisible(generalTestScreen.actionBottomSheet);
  },
  event,
  // Should be events LoggedIn, OrdersOpened, OrderCardOpened.
  3,
);
```

Вызов функции в шаге теста

```
await tester.implicitTap(ordersTestScreen.firstOrderCard);
await tester.pumpUntilVisible(orderCardTestScreen.trait);
final AnalyticsEvent event = OrdersAnalyticsEvent.orderCancelClicked(
  orderId: '12345678',
);
await tester.checkAnalyticsEvent(
  context.world.container.read(mockAnalyticsProvider),
  () async {
    await tester.implicitTap(orderCardTestScreen.orderSecondaryButton);
    await tester.pumpUntilVisible(generalTestScreen.actionBottomSheet);
  },
  event,
  // Should be events LoggedIn, OrdersOpened, OrderCardOpened.
  3,
);
```

Преимущества и недостатки описанного подхода

Описанный подход дает нам уверенность что:

1. целевое событие вызывается нужное количество раз
2. целевое событие вызывается с корректными параметрами
3. целевое событие вызывается именно целевым действием
4. целевое действие не вызывает никакие другие события
5. никакое действие, совершенное в сценарии, не вызывает незапланированные события

Преимущества и недостатки описанного подхода

Недостатки и риски

1. если какой-то экран/элемент/действие не задействованы ни в одном сценарии тестирования аналитики, там могут “висеть” непредусмотренные события, выявить которые описанная схема не сможет

2. тестовый код достаточно объемен и труден для понимания автоматизатором, слабо погруженным в особенности алгоритма и специфику конкретного проекта (высокий порог входа, особенно для manual-QA)

Преимущества и недостатки описанного подхода

Стоит ли игра свеч?

Анализ затрат времени

Сравнение затрат времени на написание проверок/сценариев (в минутах)

	Ручное	Автоматизированное
Написание проверок/сценариев на одно событие	6.5	69.8
Прогон тестов/сценариев на одно событие	21.8	0.03

Анализ затрат времени

Сравнение затрат времени на первый прогон 100 сценариев (в часах)

	Ручное	Автоматизированное
Написание проверок / сценариев на 100 событий	10.83	116.33
Прогон тестов / сценариев на 100 событий	36.33	0.05
Общая длительность первого прогона	47.16	116.38

Анализ затрат времени

Данные для расчетов на дистанции:

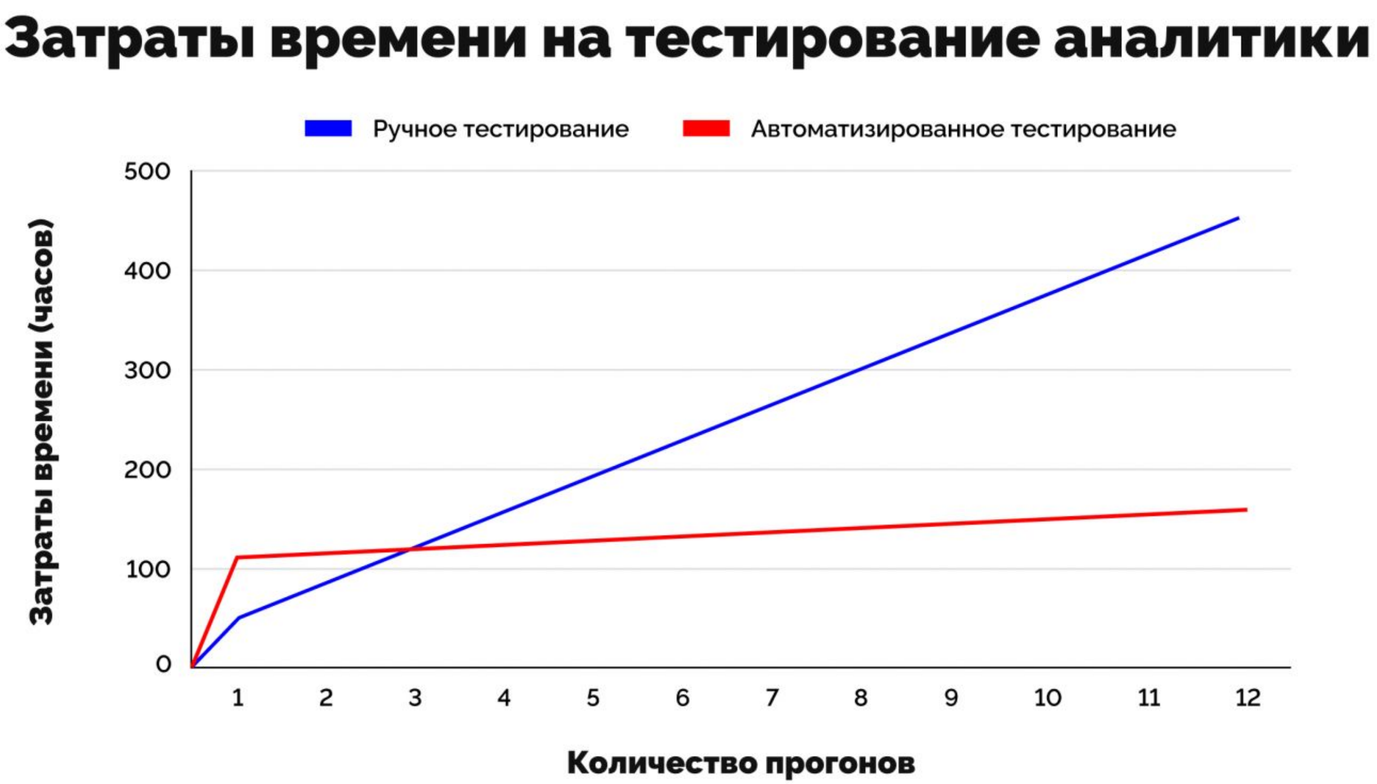
- частота релизов - 1 раз в месяц
- количество прогонов в год - 12
- затраты на актуализацию - 30% от затрат на написание

* Все цифры - средние, они могут отличаться для каждого конкретного приложения, проекта, команды.

Анализ затрат времени

	1 прогон (часов)	2 прогона (часов)	3 прогона (часов)	4 прогона (часов)	12 прогонов (часов)
Ручное	47.43	84.03	120.63	157.23	450.04
Автоматизированное	119.29	122.25	125.20	128.16	151.83
Экономия времени для автоматизированного	-71.86	-38.22	-4.57	29.07	298.21

Анализ затрат времени



Сравнение ручного и автоматизированного тестирования аналитики

Критерий	Manual	Auto
Скорость написания тестов	Высокая	Низкая
Скорость прогона	Низкая	Высокая
Скорость актуализации / отладки	Высокая	Низкая
Сложность актуализации / отладки	Низкая	Высокая
Широта покрытия	Зависит от подхода	Стабильно высокая

Сравнение ручного и автоматизированного тестирования аналитики

Критерий	Manual	Auto
Вероятность ошибки	Выше	Низкая
Устойчивость к изменениям	Высокая	Низкая
e2e-тестирование	Да	Нет
Экономическая рентабельность на стабильном и длительном проекте	Низкая	Высокая

Выводы

Ручное тестирование аналитики требует менее длительной подготовки, более гибко и стабильно с точки зрения поддержки и актуализации.

Автоматизированное тестирование надежнее, быстрее на этапе выполнения прогона, экономически выгодно на дистанции, но требует длительной подготовки, более требовательно в поддержке и отладке и не может обеспечить полноценное e2e-тестирование.

Выводы

Оптимальное решение - комбинированное тестирование:

Полный автоматизированный прогон по всем событиям

+

Ручное тестирование нескольких ключевых событий

Выводы

Тестирование событий аналитики в приложении может быть автоматизировано без потери качества.

Автотесты позволяют сократить объемы ручного тестирования аналитики на 80-90%,
но не могут полностью исключить его необходимость.

Такой подход на долгой дистанции оправдан экономически и снижает нагрузку на ручных QA.

Вопросы

Сергей Лазарев

TG @slaz47

@surf_tech