

# Критический урок: Теория ≠ Практика в разработке

---

**Версия:** 1.0.0

**Дата:** 18 июля 2025

**Контекст:** Ошибка с переключением Feed в iOS приложении






## Критическая ошибка

Была допущена грубейшая ошибка в процессе разработки: **Теоретический анализ кода был принят за подтверждение работоспособности функционала**

## Что произошло

Заявление без проверки

"Переключение работает мгновенно и надежно" - это заявление было сделано на основе:

-  Анализа кода
-  Логической корректности решения
-  БЕЗ фактического запуска
-  БЕЗ визуального подтверждения
-  БЕЗ проверки логов





Фактическая проверка показала

1. **UserDefaults работает** - значения сохраняются корректно
2. **Приложение не крашится** - запускается без ошибок
3. **НО:** Не подтверждено что новая лента фактически отображается!

## Ключевые выводы для методологии

1. Новое правило: "Код без фактической проверки = несуществующий функционал"

Даже если:

- Тесты проходят 
- Код выглядит правильно 
- Логика кажется корректной 
- Анализ показывает что "должно работать" 

**БЕЗ ФАКТИЧЕСКОГО ЗАПУСКА И ПОДТВЕРЖДЕНИЯ - функционал НЕ существует!**

2. Обязательные шаги проверки UI изменений

1. **Визуальное подтверждение**

- Скриншот до изменения
- Скриншот после изменения
- Явное сравнение

## 2. Логирование критических мест

```
init() {  
    ComprehensiveLogger.shared.log(.ui, .info, "ViewName initialized")  
}
```

## 3. UI тесты для переключений

- Тест дефолтного состояния
- Тест переключения туда
- Тест переключения обратно
- Тест сохранения состояния

## 4. Мониторинг в реальном времени

- Log server должен быть запущен
- Console logs должны отслеживаться
- Метрики должны записываться

## 3. Защита от race conditions в SwiftUI

```
// ❌ Плохо – возможна гонка  
@StateObject private var manager = Manager()  
if manager.someValue { ... }  
  
// ✅ Хорошо – прямое значение  
@AppStorage("key") private var value = false  
if value { ... }
```

## 4. Процесс верификации UI функционала

```
Шаг 1: Запустить приложение  
Шаг 2: Сделать скриншот текущего состояния  
Шаг 3: Выполнить действие  
Шаг 4: Сделать скриншот нового состояния  
Шаг 5: Проверить логи на наличие событий  
Шаг 6: Запустить UI тесты  
Шаг 7: Только после всех проверок – заявлять о работоспособности
```

## 🔧 Изменения в процессе разработки

Было (неправильно):

1. Написать код
2. Проанализировать логику
3. Заявить что "работает"

Стало (правильно):

1. Написать код
2. Добавить логирование
3. Запустить приложение
4. Визуально проверить
5. Запустить UI тесты
6. Проверить логи
7. Только потом подтверждать работоспособность



## Чек-лист для UI изменений

- ☐ Код написан
- ☐ Логирование добавлено в критические места
- ☐ Приложение успешно собрано
- ☐ Приложение запущено в симуляторе
- ☐ Скриншоты сделаны (до/после)
- ☐ Логи проверены на наличие событий
- ☐ UI тесты написаны
- ☐ UI тесты запущены и прошли
- ☐ Переключение проверено вручную
- ☐ Состояние сохраняется после перезапуска

## ⚠ Критически важно

**Любые теоретические выводы ДОЛЖНЫ перепроверяться фактическими тестами**

Это не рекомендация, а обязательное требование. При создании продукта такие ошибки недопустимы.



## Интеграция в методологию

Этот урок должен быть интегрирован в:

1. `.cursorrules` - добавить правило о фактической проверке
2. `TDD_MANDATORY_GUIDE.md` - расширить секцию о UI тестировании
3. `antipatterns.md` - добавить антипаттерн "теория вместо практики"

---

**Помните:** В разработке продукта нет места предположениям. Только факты, подтвержденные тестами и визуальными доказательствами.