



# «Простота» тестирования небольшого системного ПО

---

**Калугин Александр, PhD, PMP**

Mercury Development

Project Director



# Системное ПО



*...В отличие от прикладного программного обеспечения, системное не решает конкретные прикладные задачи, а лишь обеспечивает работу других программ, управляет аппаратными ресурсами вычислительной системы и т.д...*

© Wikipedia

# Системное ПО *на заказ*

1. «Прибанбас».
2. Небольшой компонент большого *прикладного* комплекса.
3. Реализация одного или нескольких уровней стека сетевых протоколов.
4. Клиентские/серверные компоненты комплексов использующих стандартные коммуникационные протоколы



# Почему сложнее для тестирования?

- ❑ Нету кнопок и формочек.
- ❑ Некуда вводить некорректные данные.
- ❑ Сложная предметная область.
- ❑ «Нелинейная» архитектура.
- ❑ Баг может проявляться на третьей сутки.
- ❑ То, что оно генерирует корректные данные здесь и сейчас, – ничего не значит.
- ❑ Спецификация – в виде стандарта – «слишком много букв»...



**Не всё так плохо...**



# Системное ПО / Прикладное ПО

- Мало нового кода
- Непосредственные пользователи – сервисы ОС.
- Стандартизованная логика
- Большое значение нефункциональных требований
- Сложная оптимизированная архитектура

- Много нового кода
- Непосредственные пользователи – люди
- Нестандартная логика
- Нефункциональные требования – на втором плане
- Относительно стандартная архитектура



**Просто нужен другой подход!**





# Какие дефекты типичны?

- ❑ Не ошибки в реализации «бизнес-логики», а ошибки интеграции
- ❑ Некорректные ожидания о работе сторонних компонентов и сервисов ОС
- ❑ Неправильная интерпретация стандарта
- ❑ Ошибки сложной архитектуры (нефункциональные)
- ❑ Недостаточная наработка на отказ





**Тип дефекта не специфичен  
для продукта, продукты «менее  
разнообразны»**

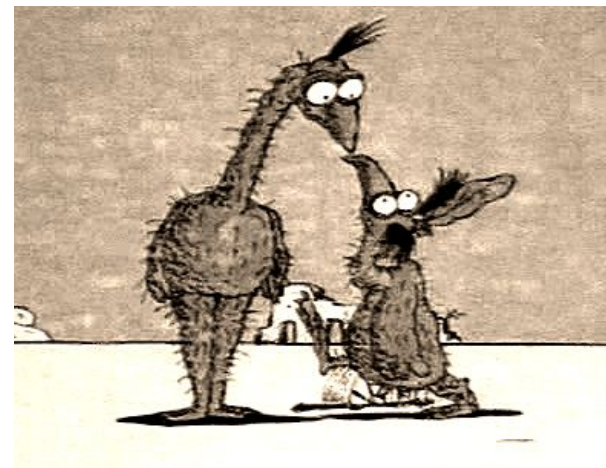


# Какие преимущества:

- ▣ Количество переходит в качество. Тестовые наборы из одного проекта могут быть использованы в другом.
- ▣ Можно подсмотреть, как играют мастера, сравнить с аналогичными продуктами, референтной реализацией
- ▣ Невизуальное представление данных – проще тестировать автоматически.



# Как тестировать?



**Прикладное ПО:** *корректное поведение - в результате анализа требований и дизайна тестовых наборов.*

**Системное ПО:** *есть референтная реализация корректного поведения аналогичным ПО.*

## **Метод №1. Сравнительное свободное тестирование**

1. Вместо анализа поведения системы на корректность и некорректность – можно сравнить с существующим аналогом.
2. При требованиях совместимости с несколькими окружениями – тестировать параллельно в различных окружениях.

# Как тестировать?



**Прикладное ПО:** *Функциональное тестирование сложной бизнес-логики. Внутреннее состояние системы – недоступно, черный ящик.*

**Системное ПО:** *Нефункциональное тестирование интеграции. Сервисы ОС поддаются конфигурированию и администрированию – серый ящик.*

## **Метод №2. Автоматизированные тесты**

1. Вместо ввода тестовых данных и креш-тестов – скриптами менять состояние системного окружения.
2. Для анализа отклика системы – не требуется специальных методов: можно анализировать (включая полуавтоматический анализ состояние сервисов используя средства анализа: `vmstat`, `tcpdump`, etc.

# Как тестировать?

**Прикладное ПО:** Для клиент-серверных систем -- нестандартные протоколы выше транспортного уровня. Клиентские и серверные компоненты – уникальны

**Системное ПО:** Используются стандартизованные протоколы, зачастую с рекомендованной реализацией



## **Метод №3. Кросс-Тестирование**

1. Вместо верификации конкретной реализации протокола – проверка совместимости работающих компонентов и поиск различий
2. Независимая проверка клиентских и серверных компонент перекрестно с использованием референтной реализации.



# Системное ПО / Прикладное ПО

- Сравнение с аналогами в полусвободном сессионном тестировании.
- Функциональное тестирование часто тривиально.
- Автоматизация достается практически бесплатно
- Возможно тестирование отдельных частей/компонентов
  
- Необходимо тестирование наработки на отказ.
- Регрессионный набор – автоматически.
- Анализ спецификации и дизайн тестов.
  
- Функциональное тестирование – центральное место
- Автоматизация – требует специальных усилий и фреймворков
- Тестирование отдельных компонентов требует специальных усилий
- Нарботка на отказ обычно не требуется.
- Регрессионный набор - специально



# Пример. NAS-клиент. Стратегия

- ❑ Разработать набор тестовых скриптов для корректности ввода-вывода (создание/удаление/перемещение файлов)
- ❑ Для каждой операции сделать дамп сетевого обмена между клиентом и сервером, используя стороннюю реализацию клиента.



- ❑ Запустить скрипт и сравнить дампы для тестируемой реализации
- ❑ Запустить скрипт в цикл – для наработки на отказ.
- ❑ Для тестирования совместимости (например, с антивирусом) – установить антивирус и повторить тест.





Спасибо за внимание! Ваши вопросы?

Калугин Александр

[alex.kalouguine@gmail.com](mailto:alex.kalouguine@gmail.com)

<http://pmarcor.com/>