МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Гжельский государственный университет»** (ГГУ)

Колледж ГГУ

Специальность 09.02.07 Информационные системы и программирование

**Реферат**

по предмету «ТРПО»

на тему «Тестовое покрытие. Оценка необходимого количества тестов»

ВЫПОЛНИЛА:

Студентка группы ИСП-О-17

Пигарева Е. А.

ПРОВЕРИЛА:

Прокуронова А. Ю.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

п. Электроизолятор

2019 год

Самый лучший способ оценить, хорошо ли мы протестировали продукт – **проанализировать пропущенные дефекты**. Те, с которыми столкнулись наши пользователи, внедренцы, бизнес. По ним можно многое оценить: что мы проверили недостаточно тщательно, каким областям продукта стоит уделить больше внимания, какой вообще процент пропусков и какова динамика его изменений.   
**Зачем оценивать?**

Любые метрики оценки – трата времени. В это время можно тестировать, заводить баги, готовить автотесты. Какую такую магическую пользу мы получаем благодаря метрикам тестового покрытия, чтобы пожертвовать временем на тестирование?  
**Поиск своих слабых зон.**Естественно, это нам нужно? не чтобы просто погоревать, а чтобы знать, где требуются улучшения. Какие функциональные области не покрыты тестами? Что мы не проверили? Где наибольшие риски пропуска ошибок?

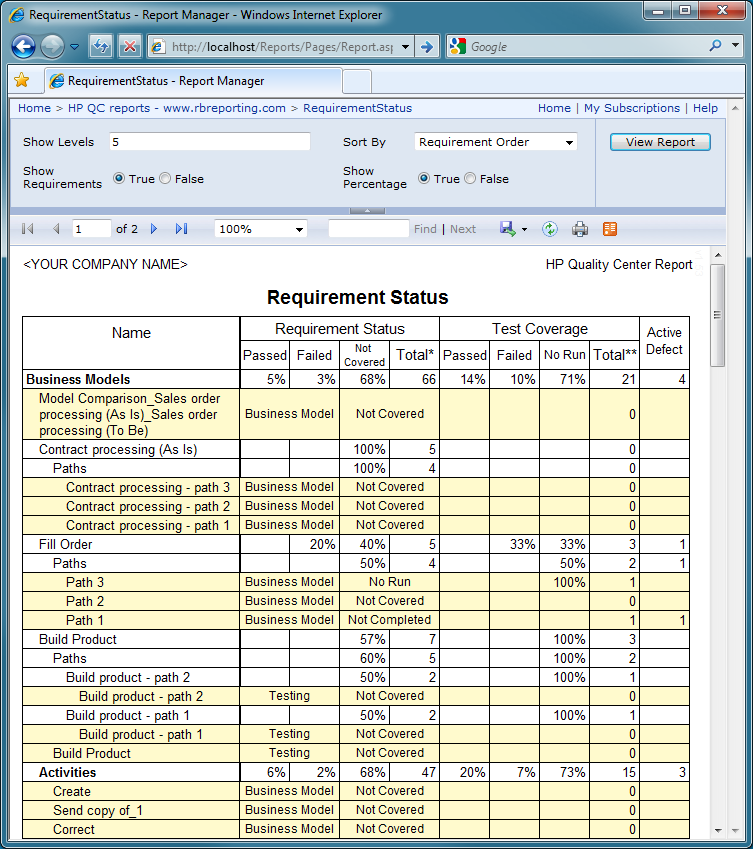
1. Редко по результатам оценки покрытия мы получаем 100%. Что улучшать? Куда идти? Какой сейчас процент? Как мы его повысим какой-либо задачей? Как быстро мы дойдём до 100?**Все эти вопросы приносят прозрачности и понятности нашему процессу**, а ответы на них даёт оценка покрытия.
2. **Фокус внимания.**Допустим, в нашем продукте около 50 различных функциональных зон. Выходит новая версия, и мы начинаем тестировать 1-ю из них, и находим там опечатки, и съехавшие на пару пикселей кнопки, и прочую мелочь… И вот время на тестирование завершено, и эта функциональность проверена детально… А остальные 50? Оценка покрытия позволяет нам приоритезировать задачи исходя из текущих реалий и сроков.

**Оцениваем покрытие требований тестами**

Допустим, у вас в команде есть аналитики, и они не зря тратят своё рабочее время. По результатам их работы созданы требования в RMS (Requirements Management System) – HP QC, MS TFS, IBM Doors, Jira (с доп. плагинами) и т.д. В эту систему они вносят требования, соответствующие [требованиям к требованиям](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%BC%D1%83_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8E#.D0.9A.D0.B0.D1.87.D0.B5.D1.81.D1.82.D0.B2.D0.BE_.D1.82.D1.80.D0.B5.D0.B1.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B9) (простите за тавтологию). Эти требования атомарны, трассируемы, конкретны… В общем, идеальные условия для тестирования. Что мы можем сделать в таком случае? При использовании скриптового подхода – связывать требования и тесты. Ведём в той же системе тесты, делаем связку требование-тест, и в любой момент можем посмотреть отчёт, по каким требованиям тесты есть, по каким – нет, когда эти тесты были пройдены, и с каким результатом.  
Получаем карту покрытия, все непокрытые требования покрываем, все счастливы и довольны, ошибок не пропускаем…  
  
**Проблема: требования не атомарны**

Аналитики тоже иногда грешат винегретом в голове, и обычно это чревато проблемами со всем проектом. Например, вы разрабатываете текстовый редактор, и у вас могут быть в системе (в числе прочих) заведены два требования: «должно поддерживаться html-форматирование» и «при открытии файла неподдерживаемого формата, должно появляться всплывающее окно с вопросом». Сколько тестов требуется для базовой проверки 1-го требования? А для 2-го? Разница в ответах, скорее всего, примерно в сто раз!!! Мы не можем сказать, что при наличии хотя бы 1-го теста по 1-му требованию, этого достаточно – а вот про 2-е, скорее всего, вполне.  
Таким образом, наличие теста на требование нам вообще ничего не гарантирует! Что значит в таком случае наша статистика покрытия? Примерно ничего! Придётся решать!

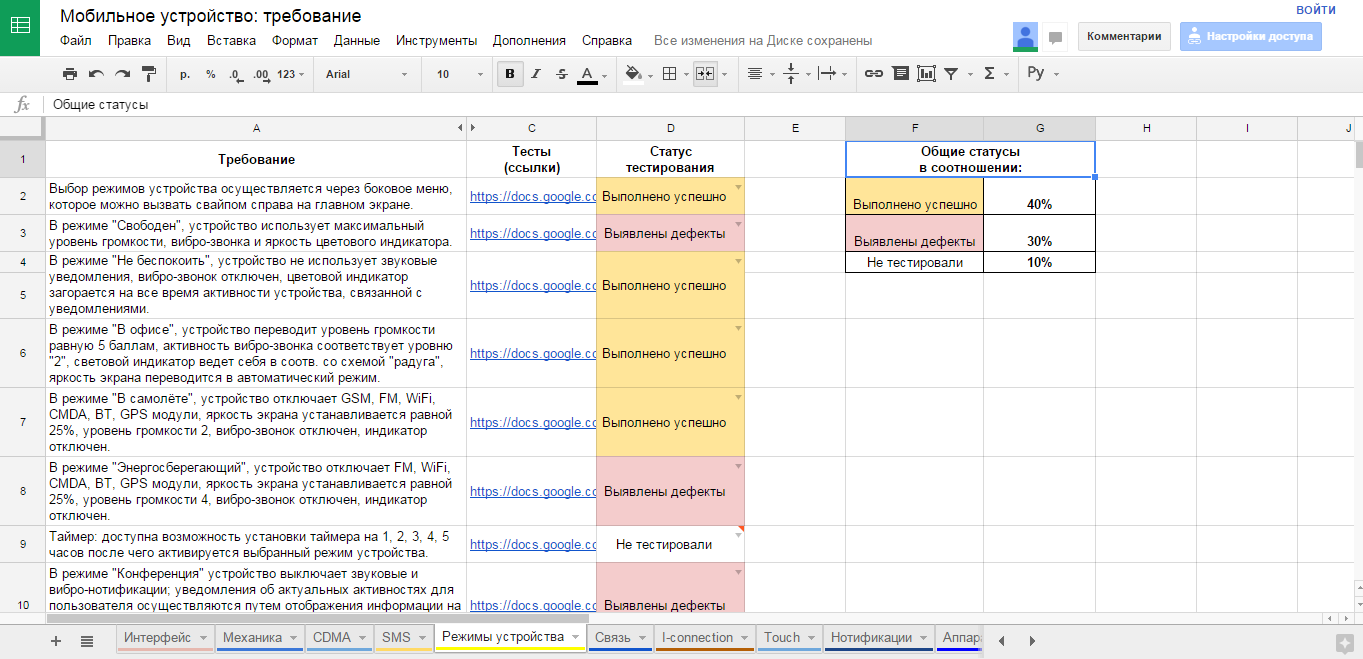
1. Автоматический расчёт покрытия требований тестами в таком случае можно убрать – он смысловой нагрузки всё равно не несёт.
2. По каждому требованию, начиная с наиболее приоритетных, готовим тесты. При подготовке анализируем, какие тесты потребуются этому требованию, сколько будет достаточно? Проводим полноценный тест-анализ, а не отмахиваемся «один тест есть, ну и ладно».
3. В зависимости от используемой системы, делаем экспорт/выгрузку тестов по требованию и… проводим тестирование этих тестов! Достаточно ли их? В идеале, конечно, такое тестирование нужно проводить с аналитиком и разработчиком этой функциональности. После доработки тестов по требованию и согласования их полноты, в системе этому требованию можно проставить статус «покрыто тестами». Эта информация будет значить значительно больше, чем «тут есть хотя бы 1 тест».

  
Конечно, такой процесс согласования требует немало ресурсов и времени, особенно поначалу, до наработки практики. Поэтому проводите по нему только высокоприоритетные требования, и новые доработки. Со временем и остальные требования подтянете, и все будут счастливы! Но… а если требований нет вообще?

**Проблема: требований нет вообще.**

Они на проекте отсутствуют, обсуждаются устно, каждый делает, что хочет/может и как он понимает. Тестируем так же. Как результат, получаем огромное количество проблем не только в тестировании и разработке, но и изначально некорректной реализации фич – хотели совсем другого! Здесь я могу посоветовать вариант «определите и задокументируйте требования сами», и даже пару раз в своей практике использовала эту стратегию, но в 99% случаев таких ресурсов в команде тестирования нет – так что пойдём значительно менее ресурсоёмким путём:

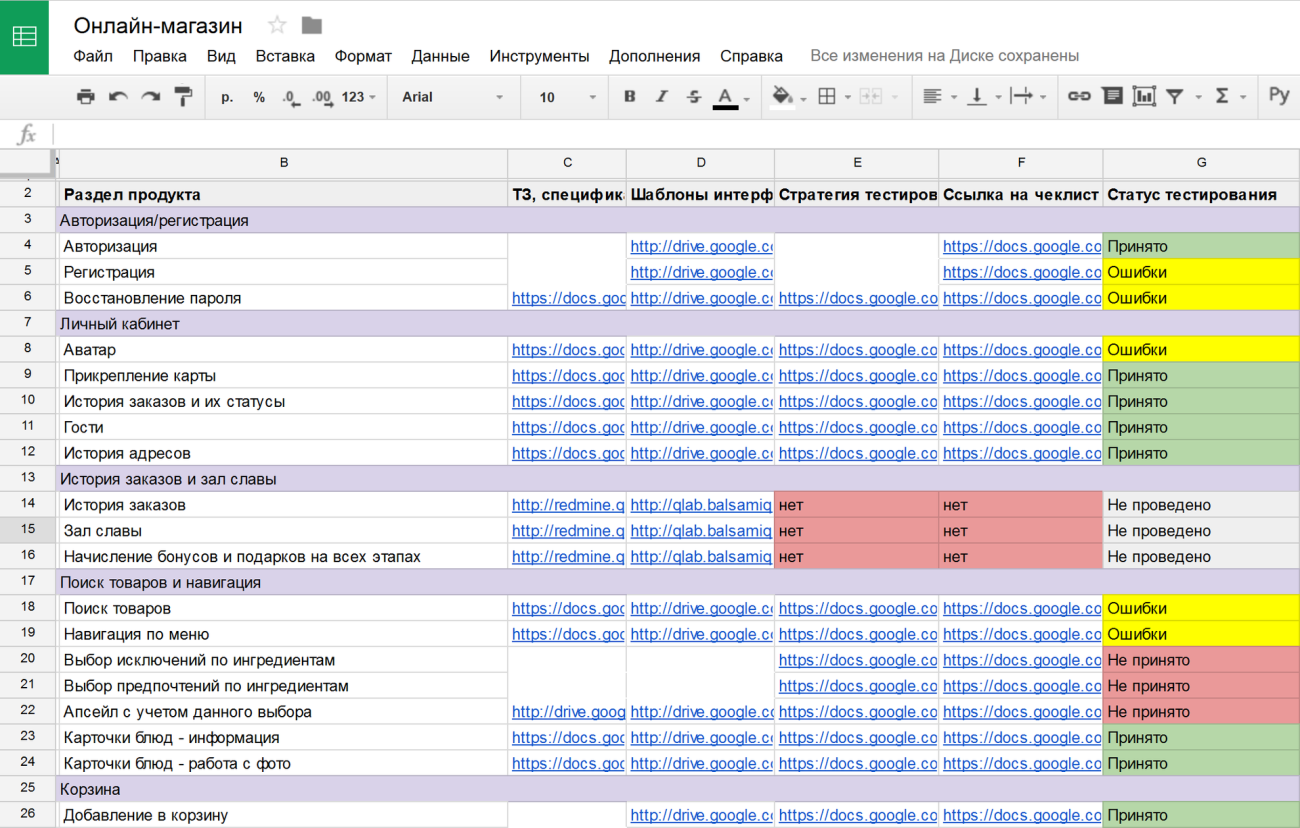
1. Создаём фичелист (feature list). Сами! В виде google-таблички, в формате PBI в TFS – выбирайте любой, лишь бы не текстовый формат. Нам ещё статусы собирать надо будет! В этот список вносим все функциональные области продукта, и постарайтесь выбрать один общий уровень декомпозиции (вы можете выписать объекты ПО, или пользовательские сценарии, или модули, или веб-страницы, или методы API, или экранные формы…) – только не всё это сразу! ОДИН формат декомпозиции, который вам проще и нагляднее всего позволит не пропустить важное.
2. Согласовываем ПОЛНОТУ этого списка с аналитиками, разработчиками, бизнесом, внутри своей команды… Постарайтесь сделать всё, чтобы не потерять важные части продукта! Насколько глубоко проводить анализ – решать вам. В моей практике всего несколько раз были продукты, на которые мы создали более 100 страниц в таблице, и это были продукты-гиганты. Чаще всего, 30-50 строк – достижимый результат для последующей тщательной обработки. В небольшой команде без выделенных тест-аналитиков большее число элементов фичелиста будет слишком сложным в поддержке.
3. После этого, идём по приоритетам, и обрабатываем каждую строку фичелиста как в описанном выше разделе с требованиями. Пишем тесты, обсуждаем, согласовываем достаточность. Помечаем статусы, по какой фиче тестов хватает. Получаем и статус, и прогресс, и расширение тестов за счёт общения с командой. Все счастливы!



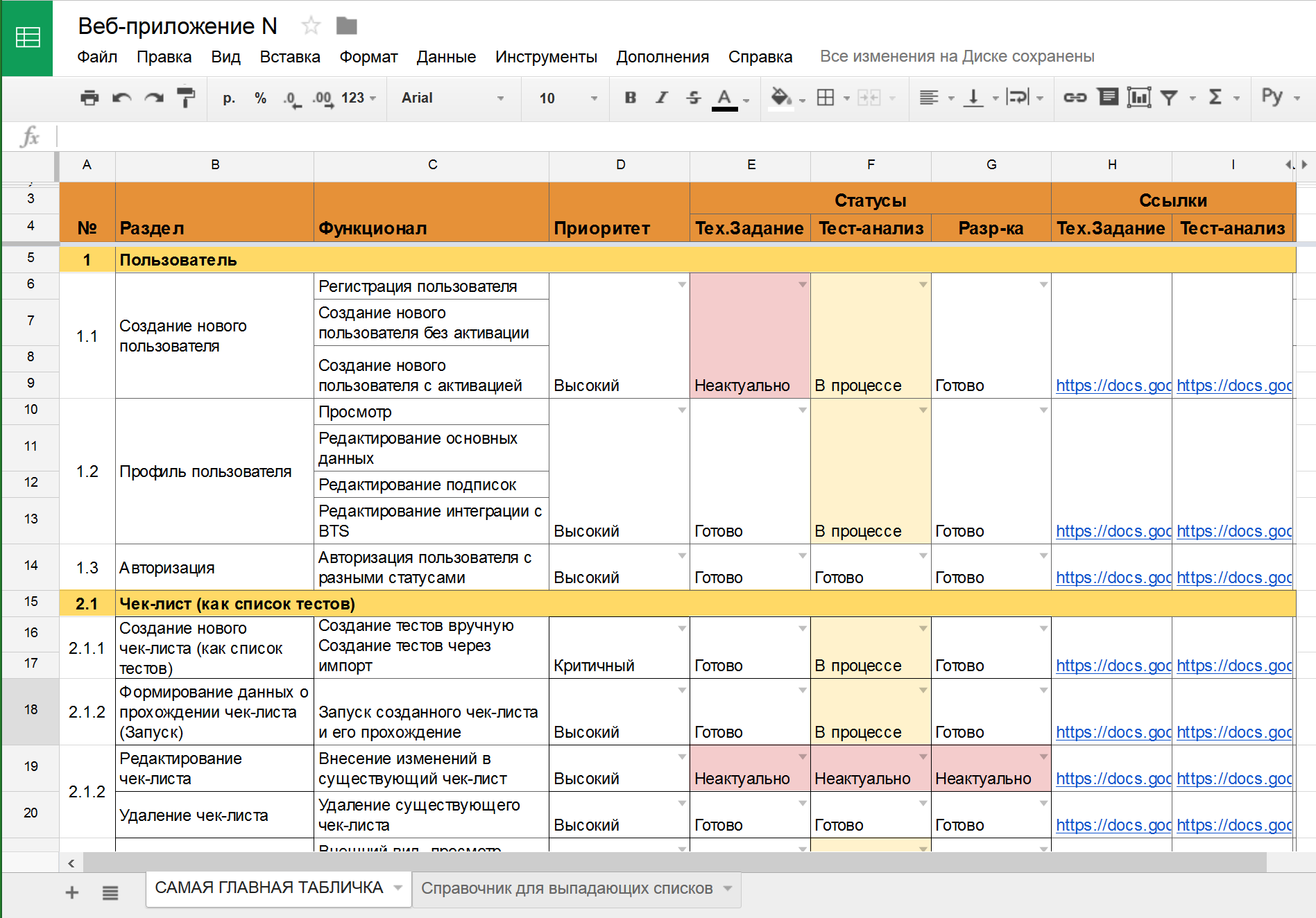
**Проблема: требования не трассируемы**

На проекте есть огромное количество документации, аналитики печатают со скоростью 400 знаков в минуту, у вас есть спецификации, ТЗ, инструкции, справки (чаще всего это происходит по просьбе заказчика), и всё это выступает в роли требований, и на проекте уже все давно запутались, где какую информацию искать?  
Повторяем предыдущий раздел, помогая всей команде навести порядок!

1. Создаём фичелист (см. выше), но без детального описания требований.
2. По каждой фиче собираем воедино ссылки на ТЗ, спецификации, инструкции, и прочие документы.
3. Идём по приоритетам, готовим тесты, согласовываем их полноту. Всё то же самое, только благодаря объединению всех документов в одну табличку повышаем простоту доступа к ним, прозрачные статусы и согласованность тестов. В итоге, у нас всё супер, и все счастливы!

  
**Проблема: требования всё время меняются**

Конечно, хорошо бы тестировать некую фиксированную систему, но наши продукты обычно живые. Что-то попросил заказчик, что-то изменилось во внешнем к нашему продукту законодательстве, а где-то аналитики нашли ошибку анализа позапрошлого года… Требования живут своей жизнью! Что же делать?



**Проблема: не хватает времени документировать тесты**

Но как оценивать покрытие тестами в таком случае?

1. Вам всё равно нужны требования, как полноценные требования или как фиче-лист, поэтому какой-то из вышеописанных разделов, в зависимости от работы аналитиков на проекте, будет всё равно необходим. Получили требования / фичелист?
2. Описываем и устно согласовываем вкратце стратегию тестирования, без документирования конкретных тестов! Эта стратегия может быть указана в столбце таблицы, на странице вики или в требовании в RMS, и она должна быть опять же согласована. В рамках этой стратегии проверки будут проводиться по-разному, но вы будете знать: когда это последний раз тестировалось и по какой стратегии? А это уже, согласитесь, тоже неплохо! И все будут счастливы.

