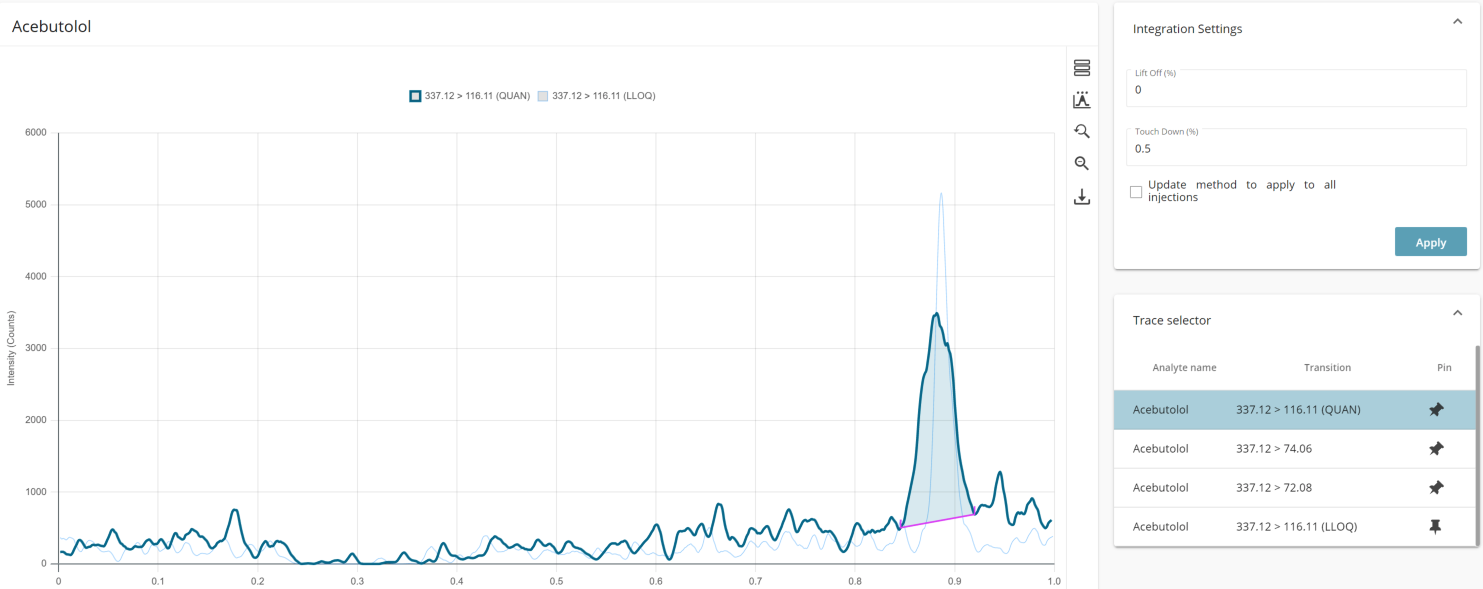
***Mind Map***

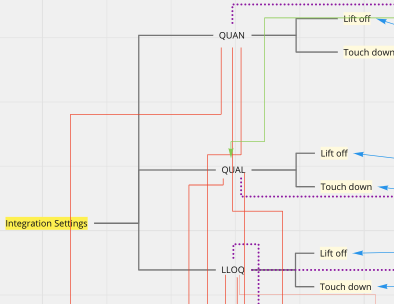
* По карте смотрим взаимосвязи - какие модули нужно протестировать (на какие влияет New Feature + провести регресс). После генерируем проверки, делаем регресс взаимосвязанных модулей.
* Для каждого приложения нужна своя Mind Map. Если в прилаге сложные взаимосвязи - то карта взаимосвязей (как у MSRQ). Если же много переходов, и функциональность меняется, то карта состояний и переходов
* Mind карта должна описывать взаимосвязи модулей, сущностей и элементов, без пояснения проверок (для этого есть Test Survey)
* Для каждого модуля строить свою карту Result Set: (Cals&QCs, Blanks, Internal Standards, Peak Integration, Unknowns). Для Method, Rule Set, Injections свои карты не нужны, так как они влияют на Result Set, а для него мы уже пропишем проверки, где Method, Rule Set, Injections уже будут (как влияющие на тот или иной модуль)
* В карте должны быть все функциональности которые зависят от других компонентов, сущностей, элементов, модулей, функциональностей. Например не надо добавлять в карту кнопку закрыть, так как она ни с чем не связана, а просто что то закрывает
* Для состояний и переходов нужна друга карта
* Если 1 элемент в функциональности может принимать разные значения (Injections: Type), то указать все значения этого элемента (которые связаны с модулем который мы составляем)
* Если связей много, то декомпозировать схему. Создать отдельную карту для каждой функциональности. А потом отдельную карту, связывающую все функциональности
* Finally, as the statistician George Box famously said, “All models are wrong; some are useful.” Don’t worry about whether or not your diagram is perfectly accurate; worry more about whether or not it gives you ideas to support your exploration.

**Оформление Mind Map**

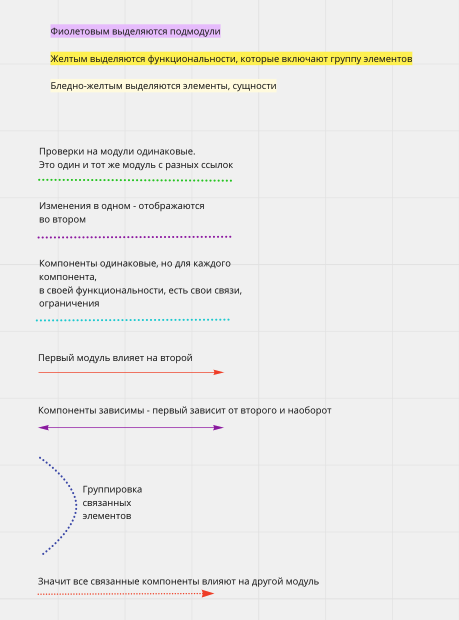
1. В Mind Map если есть 2 компонента, и нужно взаимодействие 2 сразу что бы получить результат, то вкладывать 1 модуль в другой.

Integration Settings изменяют выбранный внизу график





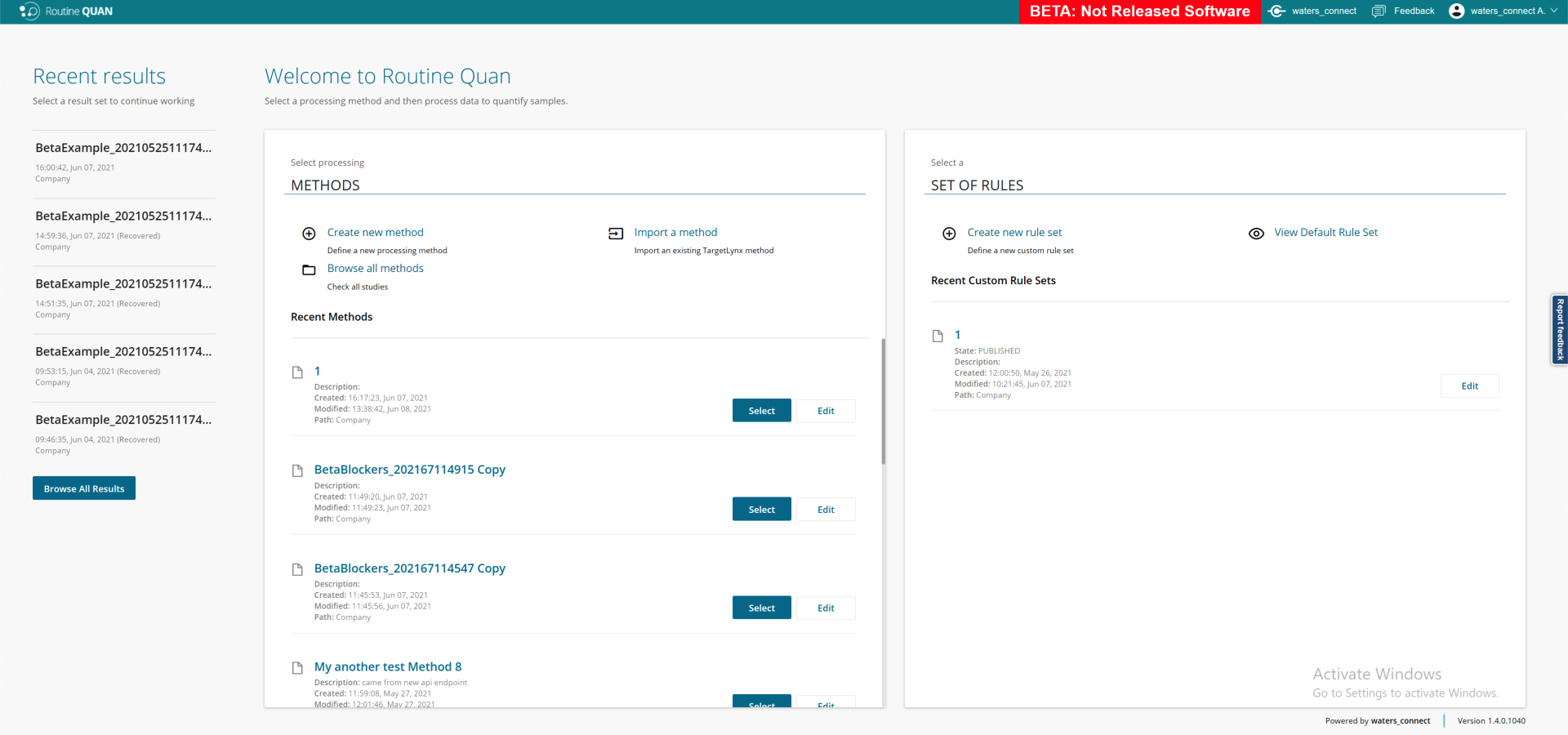
1. Стрелки взаимосвязей



***Test Survey***

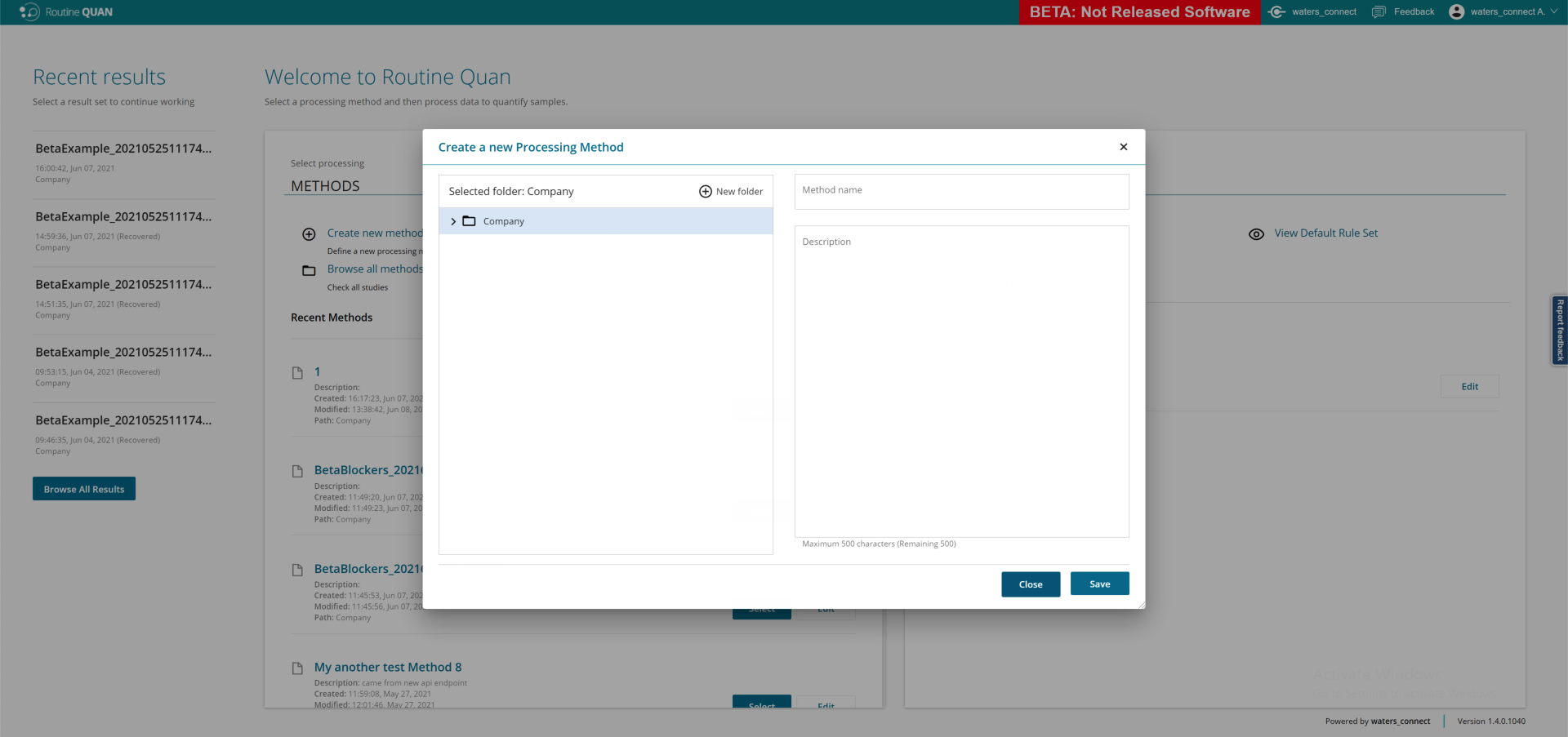
- Чем больше “глубина” приложения (модули вложены в другие модули) – тем больше будет декомпозиция по цветам

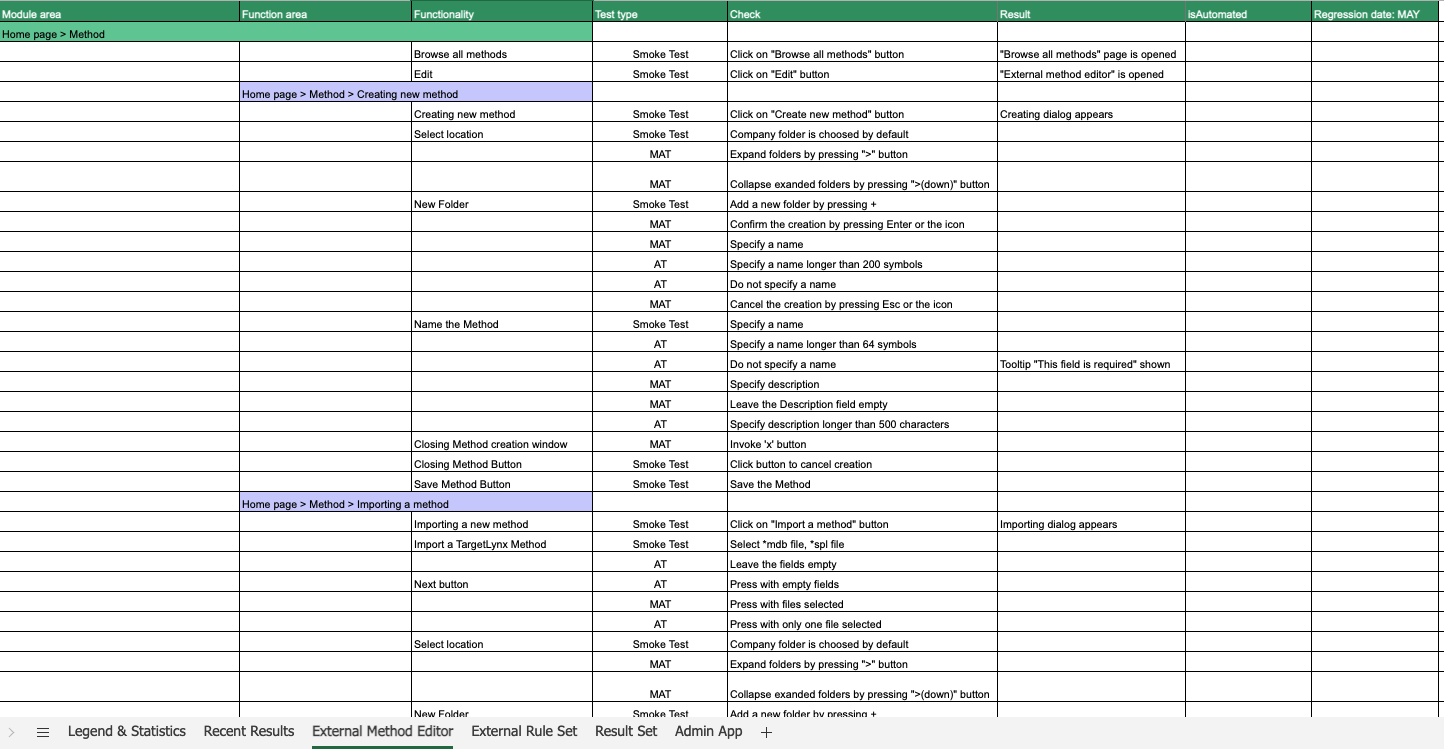
* По стандарту разбиваем приложение на Главные Модули. Каждый Главный Модуль - это отдельная страница. Но на 1 странице прилаги может быть много не связанных между собой подмодулей, функциональностей, и нет смысла декомпозировать страницу на элементы и строить карту этой страницы. Например MSRQ:Home Page - на ней расположены подмодули - Recent Results, Method, Rule Set. А при нажатии на Recent Results – открывается еще 1 модуль – Result Set. Есть смысл создать отдельные страницы для Recent Results, Result Set, Method, Rule Set



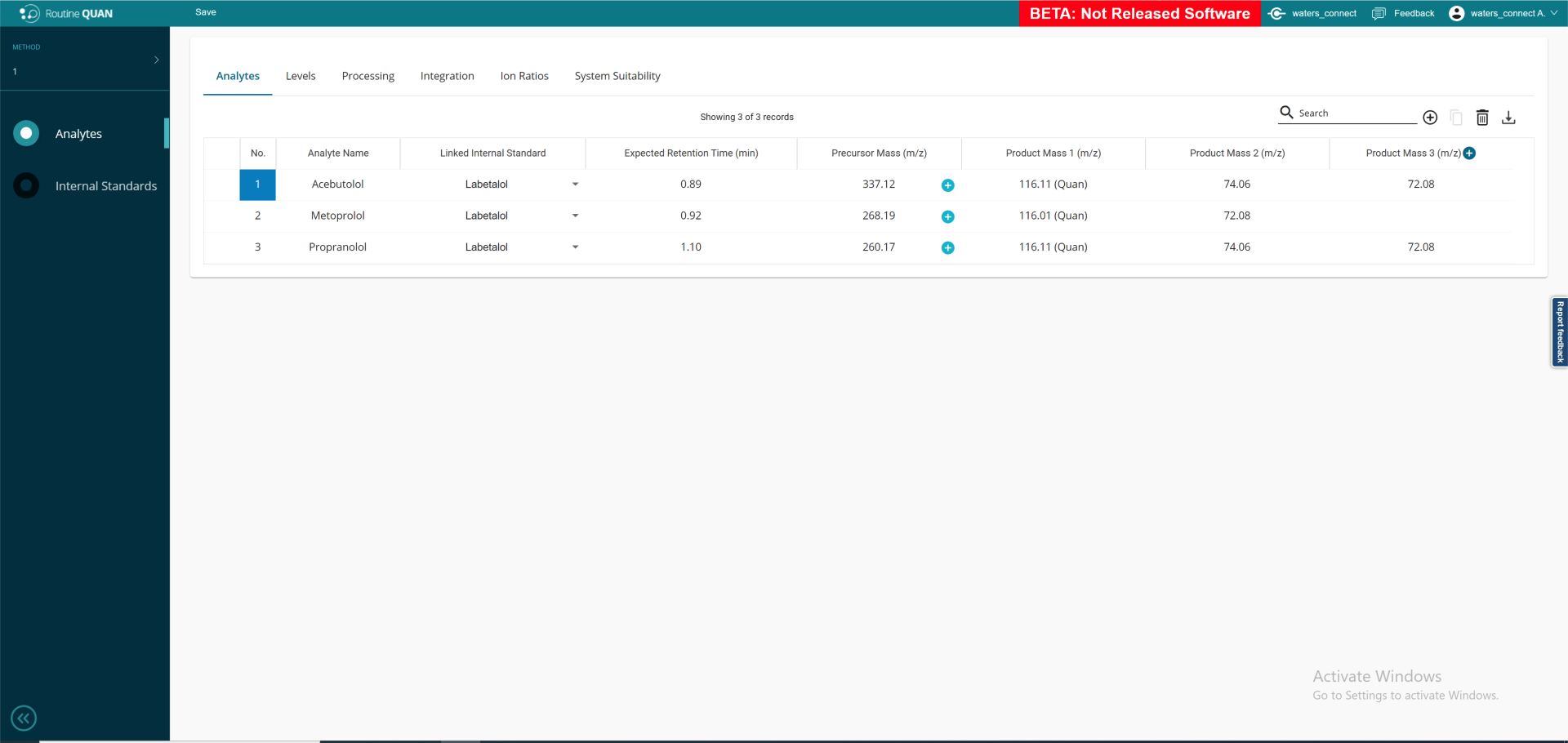


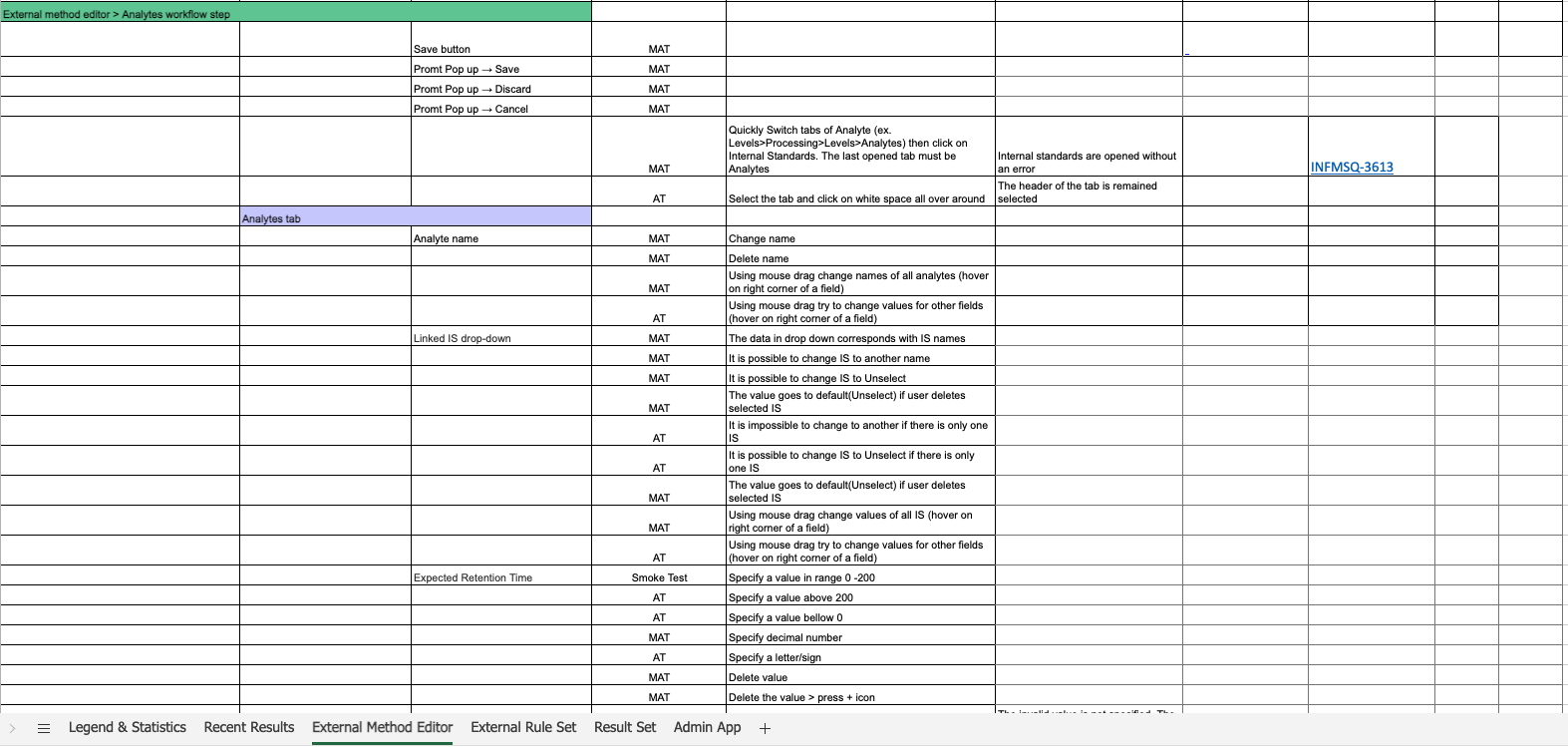
* Причем подмодуль Result Set создается путем нажатия кнопки “Select” в Methods, по этому при декомпозиции Methods – не прописывать эту кнопку. Прописать ее для подмодуля Result Set
* После декомпозируем каждый подмодуль (Recent Results, Method, Rule Set, Result Set). В test survey выделяем зеленым цветом Module Area – на какой странице располагается подмодуль
* Если открывается диалоговое окно, то Module Area не меняется, по этому новую Module Area не добавляем. Но добавляем Function Area – страница на которой располагается множество функциональностей, связанных общей логикой



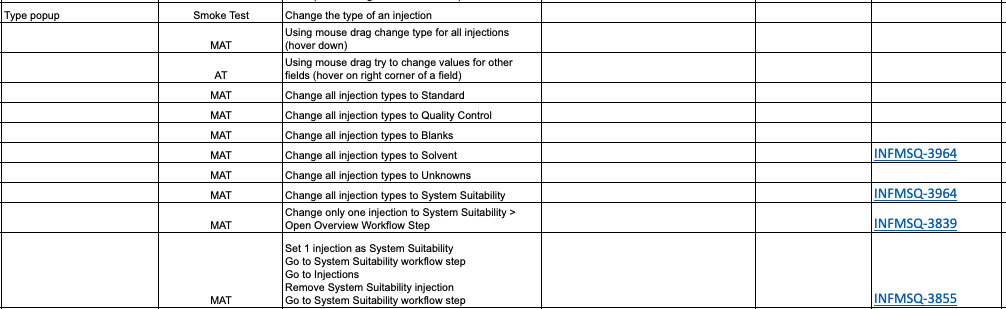


* Когда мы открываем External method editor – он располагается на другой странице. По этому выделяем в отдельный Module Area.





* Когда декомпозировали уже до функциональностей, их элементов, сущностей, ограничений (требований) – прописываем для них проверки
* Что бы смотреть взаимосвязи и проверки которые стоит дописать - профильтровать все заведенные дефекты, особенно заведенные PO или BA. После можно проверить не воспроизводится ли этот дефект в другой части приложения, так как дефекты имеют свойство находиться рядом друг с другом.Так мы проверяем на парадокс пестицидов



* Если есть вопрос за что отвечает та или иная функциональность, то создать фильтр в Jira – и посмотреть Story, в которой была реализована данная фича
* По Test Survey смотреть сколько раз встречаются дефекты и в каких модулях. Проверять этот модуль тщательнее. Так как мы линкуем дефекты - то можно отследить из за чего дефекты возникают в этих модулях – Defect clustering. Так можно увидеть какие функциональности с «Дырами» и проверить эти функциональности в других модулях, и если там не воспроизводятся дефекты, то можно подумать из за чего они воспроизводятся в первом модуле. Если это связано с тем что мы не увидели конфликты функциональностей, то выписать эти конфликты в отдельный документ
* Когда мы добавляем подстроку к строку в таблице > выделяем другую строку > удаляем ее (удаляется другая строка)
* Когда мы убрали фильтр на показ True инъекций > делаем инъекцию True (сразу несколько инъекций становится true)
* Стараться не сильно разбивать модуль на функциональности. Если при изменении какого то компонента - меняется другой (изменили значения - график стал желтым). То не добавлять в test survey новый компонент (“Желтый график”) – а после писать проверки когда график уже желтый. Нет, надо прописать проверки в те компоненты, которые делают график желтым. А если нужна проверка, которая производится после того как график стал желтым, то писать сложную проверку – “Ввести значения > нажать кнопку “применить” > (здесь писать проверку когда график стал желтым, например, ввести значения заново)

Если функциональность 2 появляется после изменений в функциональности 1, то прописывать проверки в функциональность 2, как сложные проверки функциональности 1

* Если одна функциональность изменяет вторую, то писать проверки под эти 2 функциональности (что первая меняет вторую, а во второй функциональности написать условия при которых она изменится). Не надо сокращать проверки, даже если они излишни. Так как если нужно будет протестировать только 1 функциональность, а проверок не будет прописано, то мы можем не вспомнить что на нее влияет и пропустить проверки
* Функциональность может быть активной и пассивной

Активная - это действие над 1 функциональностью. Пассивная - это эквивалентный класс 2 функциональности

*Civilization 6*

*Варвары могут захватить строителя - Активная*

*Строители могут быть захваченными варварами - Пассивная*

Проверки прописывать для двух функциональностей, хоть они и будут повторяться, но относиться будут к разным функциональностям

* Проверки делать так же на состояние элементов, сущностей - по дефолту, при изменении (например что становится с кнопкой). Если у компонентов есть своя логика по дефолту (QUAN trace уникален для каждой инъекции, а LLOQ trace 1 на все инъекции) - то это прописать отдельными проверками (больше как памятка для нас)
* После масштабной регрессии всей прилаги, составить отчёт - сколько проверок было сделано, какие дефекты были найдены, в каких модулях, какое покрытие у них было (Smoke, MAT, AT). На основе этой инфы показать в каких модулях у нас больше всего дефектов. Во время второй полной регрессии составить такой же отчёт, но ещё записать все эпики которые были сделаны. Так можно посмотреть какие эпики что сломали
* Писать проверку на то чего в модуле нет (часто модули похожи, и в 1 модуле может не быть функциональностей второго). Стоит прописывать это как отдельной проверкой, так как после не вспомнится что нужно проверять ОТСУТСТВИЕ чего то
* Если проверка находит дефект (например перейти на вкладку 1 в модуле 1 и нажать на Home Page - падает дефект), то проверить разные ее вариации (перейти на вкладку 2 в модуле 1 и нажать на `home Page, перейти на вкладку 2 в модуле 1 > включить фильтры > нажать home page, …)
* Анализировать Test Survey - что бы найти скопление дефектов. Что бы проверить воспроизводиться ли этот дефект в другом модуле. По этой схеме мы находим новые дефекты, опираясь на уже найденные дефекты.
* Анализировать Jira - что бы найти общее у всех дефектов. На проекте MS Quan 60% дефектов связаны с отсутствии чего то (отсутствует хроматограмма, тип инъекции, сама инъекция и тд). Не анализировать дефекты типа Story, так как эти дефекты мы заводим во время тестирования самой задачи, и их мы не пропускаем). По этой схеме мы определяем универсальные проверки, которые будут всегда в приданое находить баги. Эти проверки уникальны для каждого проекта

Когда нашли эти уникальные эвристики, то применить их к каждой функциональности которая есть (для каких то модулей могут отсутствовать эти проверки в Test Survey). Тем самым мы закроем все модули

Можно вынести вывод, что SMOKE проверки отваливаются только после внедрения новых модулей. Значит можно проводить полный регрессное каждый день, но только по SMOKE проверкам.

**Story Checks**

* Не обязательно заполнять все поля. Может быть такое что condition предусловия для выполнения функциональности (например указание dropline) – это что бы пик, где сам dropline будет указан - существовал. Нет смысла писать эквивалентный класс “существования” сущности “пик”, так как мы это уже сделали в condition предусловия
* Если AT эквивалентный класс не проходит в проверке, то не пройдут и другие комбинации с этим классом. Перестать это проверять
* Тестируем каждое значение из эквивалентного класса 1 раз.
* Тестировать на разных входных данных. Например на разных датасетах
* Сущность можно быть разной. Как та что существует во внешнем мире (промокод, Emal, номер телефона,…). Так и та которая создается в приложении (Аккаунт, пароль, Инъекции,…). Для создаваемых сущностей рисовать диаграмму состояний и переходов.
  + (Пароля нет - Создан - Поменян)
  + (Аккаунта нет - Создан - Удалён)
  + В каждом состоянии сущности проводить все проверки повторно
* Если тестируем невидимую подфункциональность - (отмена вставки в таблицу по ctrl z) - то проверить главную функциональность (таблица) и ее подфункциональности. А так же подумать есть ли зависимые функциональности (сделать изменения -> сделать другие изменения -> отменить первые изменения)
* При выделении перспективы мы должны понять на каком уровне мы находимся. (влияет ли функциональность на сам модуль (или на его общую сущность - например сам Result Set), на подмодуль или на функциональность)