



Александр Алехин

Базовый курс разработки по промышленным стандартам на 1С

Содержание

[Содержание](#)

[Введение](#)

[О курсе](#)

[Инфраструктура](#)

[Настройка окружения](#)

[О теме](#)

[GIT](#)

[SourceTree](#)

[OneScript & Precommit1C](#)

[BDDEditor & Behavior](#)

[Allure & IrfanView & VSC](#)

[Итоги](#)

[Домашнее задание](#)

[BDD](#)

[О теме](#)

[*.feature](#)

[Обработка проверки поведения](#)

[Состояние шага](#)

[Программирование шагов](#)

[Allure](#)

[Итоги](#)

[Домашнее задание](#)

[Организация процесса](#)

[О теме](#)

[“Лоскутная” автоматизация](#)

[Промышленные стандарты](#)

[Итоги](#)

[Домашнее задание](#)

[Практика применения](#)

[О теме](#)

[Автоматизированное тестирование](#)

[Фичи из “воздуха”](#)

[Конфигурация на поддержке или базовая](#)

[Конфигурация с возможностью изменения](#)

[Итоги](#)

[Финальное задание](#)

Введение

О курсе

Коллеги, здравствуйте! Меня зовут Александр Алехин. Я работаю в команде “Серебряная пуля”. В этом курсе вы узнаете как внедрить в вашей команде разработку по промышленным стандартам, т.е. вы научитесь настраивать окружение разработчика, эффективно организовывать процесс на всех этапах работы, а так же использовать соответствующие инструменты.

Инфраструктура

Для записи этого видео-курса, я буду использовать виртуальную машину под управлением операционной системы Microsoft Windows Seven. Так же, на эту виртуальную машину я установил 1С:Предприятие версии 8.3.8 и Microsoft .Net Framework версии 4.5. Ещё мне понадобятся: браузер, я выбрал google chrome и архиватор, в моем случае это winrar.

Настройка окружения

О теме

В этой теме мы сформируем окружение разработчика. Кроме самой платформы 1С у нас появиться система контроля версий и удобный клиент для работы с ней. Мы познакомимся с инструментом для разбора внешних отчетов и обработок 1С на исходные файлы. Настроим инструменты для выполнения сценариев пользовательских действий, формирования наглядного отчета о проверки поведения, создания авто-инструкций. А так же познакомимся с социальной сетью для разработчиков. В общем тема важная и очень интересная, давайте начинать.

GIT

Формировать окружение разработчика начнем с системы контроля версий и использовать мы будем GIT. Зайдем в браузер, в строке поиска введем

git

и выполним поиск. Перейдем на страницу с адресом <https://git-scm.com/> и нажмем на кнопку Download for windows. Если автоматически не началась загрузка то вы можете запустить ее самостоятельно нажав на соответствующую вашей операционной системе и её разрядности ссылку 32-х или 64-х битную. Пока выполняется загрузка, покажу вам как найти документацию git на русском языке. Нужно перейти по ссылкам Documentation → Book → Русский. Дистрибутив скачался, запускаем установку и идем по шагам:

Шаг	Описание	Действие
1	Предлагается ознакомиться с лицензионным соглашением	Читаем и жмем кнопку Next
2	Дается возможность выбрать каталог в который будет установлен git	Оставим без изменения и перейдем на следующий шаг
3	Можем изменить состав компонент для установки	Оставим без изменения и перейдем на следующий шаг
4	Можем настроить создание ярлыков	Оставим без изменения и перейдем на следующий шаг
5	Должны выбрать вариант использования переменной	Выберем третий вариант, который нам позволит использовать из командной строки

	среды "Path" (чтобы увидеть и отредактировать данную переменную можно зайти в Свойства компьютера → Дополнительные параметры системы → Переменные среды → Системные переменные → Path)	не только GIT но и дополнительные инструменты Unix
6	Предлагается изменить настройки преобразования строки	Оставим без изменения и перейдем на следующий шаг
7	Предлагается изменить настройки эмулятора терминала	Оставим без изменения и перейдем на следующий шаг
8	Предлагается изменить настройки дополнительных опций	Оставим без изменения и нажмем кнопку Finish

Есть два способа работы с GIT, через консоль и через sourcetree. В рамках курса работать с GIT мы будем через sourcetree, но основные моменты работы с GIT через консоль мы все же рассмотрим. Для этого создадим каталог в котором будем держать все наши репозитории, желательно в корне, в моем случае это будет диск "C:\repo". А в нем я создам каталог "work" который и будет нашим репозиторием.

К слову сказать "Репозиторий" это обычный каталог windows который находится под контролем той или иной системы контроля версий, в нашем случае GIT. Вызовем контекстного меню проводника и выберем пункт "Git Bash Here" и напишем

```
git init
```

и нажмем клавишу Enter чтобы выполнялась команда. Мы получили сообщение о том что был инициализирован пустой git репозиторий, но в нашем каталоге мы не видим никаких изменений. Дело в том что по умолчанию системный каталог не видимы, чтобы это исправить нажмем Alt → Сервис → Параметры папок → Перейдем на закладку "Вид" → И в самом конце списка дополнительных параметров выберем вариант "Показывать скрытые файлы, папки и диски" → Нажмем Ок и увидим системный каталог .git (пока ничего с ним делать не будем).

Далее нам нужно выполнить настройки нового репозитория, укажем имя пользователя, для этого выполним команду

```
git config --local user.name "primer"
```

укажем электронный адрес этого пользователя, для этого выполним команду

```
git config --local user.email priner@primer.com
```

чтобы проверить указанные данные выполним команду

```
git config --list
```

Мы видим что имя и электронный адрес были успешно зафиксированы в локальном конфигурационном файле. Так же эти данные мы можем увидеть в каталоге .git в файле .config.

Теперь, давайте на простом примере посмотрим как работает git, выполним команду

```
git status
```

Видим что сейчас мы находимся на ветке master и нет ни каких изменений которые можно было бы сохранить. Создадим в репозитории текстовый файл с именем primer и добавим в него строку с текстом

```
Строка 1
```

Чтобы посмотреть изменения выполним команду

```
git status
```

Получаем сообщение о том что GIT в своем репозитории видит файл, но изменения самого файла он не контролирует. Чтобы GIT взял файл под свой контроль выполним команду

```
git add .
```

Теперь мы можем продолжить работу с файлом, т.е. открыть его в блокноте и что-то подредактировать, а можем прямо сейчас сохранить его состояние выполнив команду

```
git commit -a -m "Создал и добавил строку"
```

Снова выполним команду

```
git status
```

И увидим что в репозитории нет не сохраненных изменений. Давай те изменим файл, например: добавим строку с текстом

```
Строка 2
```

Снова выполним команду

```
git status
```

Получаем сообщение о том что есть измененный файл, но сохранить этой файл командой commit мы сможем только после того как проиндексируем этот файл. Для этого выполним команду

```
git add .
```

Т.е. перед тем как сохранить новый файл или уже существующий нужно его проиндексировать, теперь когда мы это сделали - сохраним, а точнее говоря “закоммитим” изменения командой

```
git commit -a -m "Добавил строку"
```

И тут же, чтобы посмотреть изменения выполним команду

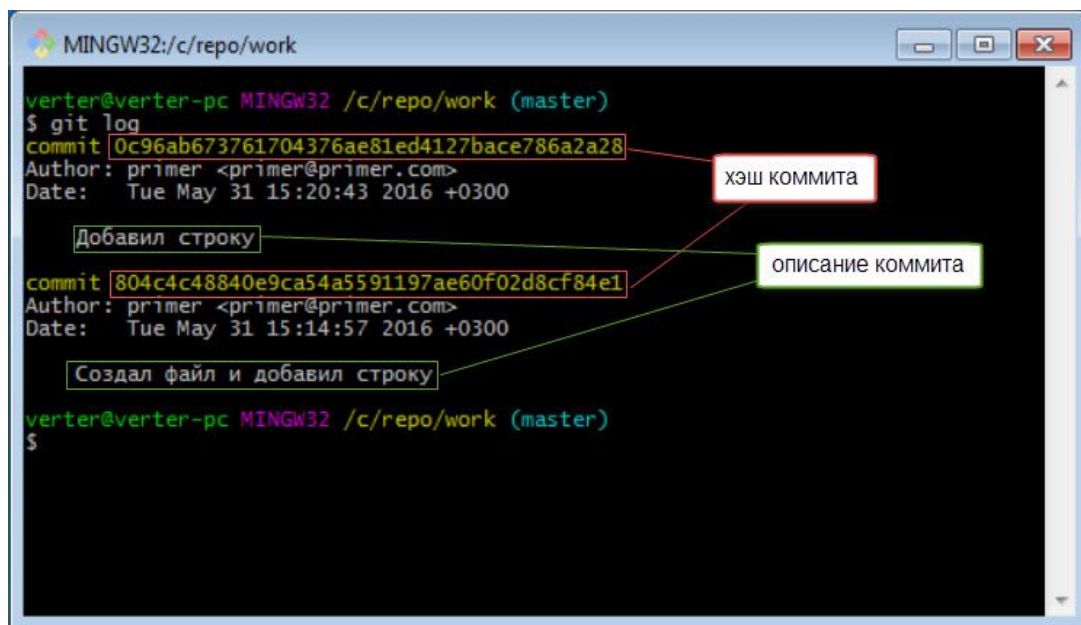
```
git status
```

Да, не сохраненных изменений нет.

Теперь пришло время показать зачем мы выполняли все эти команды. Во-первых у нас теперь есть история изменения файлов, мы можем ее посмотреть с помощью команды

```
git log
```

Выполним ее и увидим все сделанные коммиты.



The screenshot shows a terminal window titled 'MINGW32:/c/repo/work'. The user runs 'git log', which displays two commit entries. The first commit has a hash '0c96ab673761704376ae81ed4127bace786a2a28' and the message 'Добавил строку'. The second commit has a hash '804c4c48840e9ca54a5591197ae60f02d8cf84e1' and the message 'Создал файл и добавил строку'. Red and green boxes highlight the hashes and messages, with labels 'ХЭШ КОММИТА' and 'описание коммита' pointing to them. The terminal prompt is '\$'.

Во-вторых мы можем “вернуться в прошлое”, т.е. откатить состояние репозиторий к какому-либо коммиту, для этого скопируем хэш коммита “Создал файл и строку” и выполним команд

```
git checkout <хэш коммита>
```

Откроем наш файл и увидим что из него пропала “Строка 2” которую мы добавляли после этого коммита. Ну и в-третьих мы можем вернуться к самому последнему состоянию репозитория. Чтобы увидеть самый последний по времени коммит выполнив команду

```
git log --all
```

Скопируем его хэш и выполним команду

```
git checkout <хэш последнего по времени коммита>
```

Откроем файл и увидим вторую строку.

Итак, вариант работы с GIT через консоль мы рассмотрели, а чтобы изучить второй вариант - нужно познакомиться с sourcetree.

SourceTree

Продолжаем формировать окружение разработчика и следующий инструмент - sourcetree. SourceTree - это русифицированный клиент GIT, который дает нам возможность работать с GIT не через командную строку, т.е. не через консоль, а через пользовательский интерфейс. Для работы с sourcetree кроме самого приложения нам потребуется аккаунт на bitbucket. К слову сказать bitbucket - это веб-сервис для хостинга

репозитории и совместной разработки, но об этом мы поговорим позднее и в сравнении с его конкурентом github. Сейчас займемся тем что создадим аккаунт на bitbucket. В поисковой строке браузера напишем

atlassian

и выполним поиск. Дело в том что компания Atlassian является владельцем как bitbucket так и sourcetree, т.е. через нее мы выйдем на оба продукта. Перейдем по ссылке <https://ru.atlassian.com/> и далее по ссылке: Продукты → Все продукты → Вот мы видим в одной строке и bitbucket и sourcetree. Начнем с bitbucket, кликнем по соответствующей ссылке. Нажмем на кнопку “Начните работу бесплатно”. Мы перешли на сайт bitbucket и чтобы зарегистрироваться кликнем на ссылку Get started. Укажем реальный адрес своей электронной почты и нажмем кнопку “continue”. Укажем свое полное имя и пароль. Ещё от нас потребуется подтвердить свою “человечность”, снова нажмем “continue” и теперь чтобы завершить регистрацию нам осталось подтвердить адрес своей электронной почты, для этого перейдем в почту, найдем письмо от atlassian и нажмем на кнопку подтверждения. Итак, у нас есть аккаунт, переходим к скачиванию и установке sourcetree.

Вернемся на страницу продуктов atlassian и выберем sourcetree. В открывшейся странице нажмем на кнопку “download for windows” чтобы началось скачивание дистрибутива. Итак, дистрибутив скачался кликнем по нему чтобы начать установку и нажмем кнопку “запустить”.

Шаг	Описание	Действие
1	Приветствие	Жмем кнопку Next
2	Можем изменить каталог в который будет установлен sourcetree	Оставим без изменения и перейдем на следующий шаг
3	Запуск установки	Жмем кнопку Install
4	Финальный	Жмем кнопку Finish

При первом старте от нас потребуется выполнить некоторые настройки и пройти авторизацию.

Шаг	Описание	Действие
1	Лицензионное соглашение	Соглашаемся и переходим на следующий шаг
2	Нужно авторизоваться с помощью аккаунта на bitbucket	Жмем на кнопку use an existing account чтобы воспользоваться существующим аккаунтом. В появившемся окне вводим данные

		авторизации на bitbucket и переходим на следующий шаг.
3	Можем подключиться к серверу	Оставим без изменения и перейдем на следующий шаг

SSH ключа у нас нет, поэтому на вопрос мы ответим отрицательно, а в следующем сообщении выберем вариант “Я не хочу использовать Mercurial”. Готово - окно sourcetree открылось.

Теперь давайте все что мы сделали с GIT через консоль - воспроизведем с помощью sourcetree. Для начала создадим репозиторий, для этого нажмем на кнопку “Клонировать/Создать”. В появившемся окне перейдем на закладку “Создать новый репозиторий” и в поле “Целевой путь” укажем путь к новому репозиторию, вспоминаем что у нас есть целый каталог в котором мы держим все наши репозитории (это каталог “hero”) и в нем добавим ещё один, например “work1” и нажмем на кнопку “Создать”. Давайте посмотрим что физически произошло с репозиторием work1. Для этого выделим в дереве подключенных к sourcetree репозиториев наш репозиторий, вызовем контекстное меню и выберем пункт “Посмотреть в проводнике”. Да, мы видим специальный, а точнее сказать системный каталог .git. На следующем шаге, нам нужно выполнить настройки репозитория, т.е. нам нужно выполнить какие-то действия в sourcetree в результате которых в системном каталоге .git в файле config появятся данные пользователя, давайте откроем этот файл и проверим что данных о пользователе в нем сейчас нет. Вернемся в sourcetree и выполним пункты меню: Репозиторий → Настройки репозитория → И перейдем на закладку “Дополнительно” → Снимем флаг “Использовать глобальные настройки пользователя” (об этом мы ещё поговорим) → И введем имя пользователя и адрес его электронной почты. Посмотрим файл config → Да, данные пользователя появились.

Теперь давайте напloedим коммитов. Для этого создадим в репозитории текстовый файл, например “text” и вернемся в sourcetree. Ожидаем что sourcetree подхватит и отобразит сделанные нами изменения, если этого не произошло - можно нажать F5. Да, sourcetree на закладке “Состояние файлов” отобразил изменения состояния репозитория в списке файлов вне индекса, как вы уже поняли нам нужно установить флажки на против тех файлов, которые мы хотим проиндексировать. Установим флаг и увидим, что наш файл перешел в список проиндексированных файлов. Теперь нажмем на кнопку “Закоммитить” (в верхней части окна) и добавим описание выполненных действий, в описании коммита нужно по честному отразить то что было сделано, а чтобы не писать “мемуары” коммитить нужно чаще - это вам позволит точнее перемещаться во времени, т.е. по коммитам. Итак, напомним “Создал файл” и нажмем на кнопку “Закоммитить” (в нижней части окна). Перейдем на закладку “Журнал/История”, чтобы увидеть историю коммитов. Откроем файл в редакторе и добавим строку

Вернемся в sourcetree и на закладке “Журнал/История” в списке коммитов должны появиться строка с текстом “Незаконмиченные изменения”. Встанем на нее и выполним коммит, закрепим, добавим ещё строку в файл

Строка 2

индексируем нужные файлы, жмем кнопку “Закоммитить” (в верхней части окна), добавляем описание “Добавил строку”, жмем кнопку “Закоммитить” (в нижней части окна). Добавим ещё строку и закоммитим. Итак в нашей истории уже три коммита. Перемещаясь по строкам в списке коммитов (**кликаая по ним один раз**) мы можем просматривать данные коммита, т.е. его автора и описание, список измененных файлов а так же что именно изменилось в файле. Уже не плохо правда? НО мы ведь ещё не путешествовали по коммитам. Переключиться на тот или иной коммит можно **кликнув по нему в списке коммитов дважды**. Сейчас в нашем файле две строки, если мы перейдем на первый коммит, то в файле не должно быть ни одной строки. Кликнем дважды по самому раннему коммиту, получаем окно предупреждения, то о чем идет речь в этом предупреждении нас не касается, т.к. мы вернемся на последний по времени коммит и все встанет на свои места, добавлю лишь что это предупреждение связано с ветвлением, но о нем мы пока говорить не будем, просто нажмем “Ок”. Посмотрим файл, строк нет, т.е. состояние репозитория и нашего файла в частности, было приведено к состоянию на момент первого коммита. Теперь вернемся к последнему по времени коммиту и убедимся что файл у нас вернулся к своему конечному состоянию. Кстати, то что первый наш репозиторий был создан через консоль не означает что с ним нельзя работать через sourcetree. Чтобы это проверить - нажмем на кнопку “Клонировать/Создать” → Перейдем на закладку “Добавить рабочую копию” → В поле “Путь к рабочей копии” выберем репозиторий work и нажмем кнопку “Добавить”. Все что нам остается сделать - это дважды кликнуть по ветке master. Подводя итоги можно сказать что работать через sourcetree с git удобней и наглядней, только нужно учитывать что sourcetree отправляет команды в git не только по нажатию на какую либо кнопку но и по одинарному и двойному клику. Итак мы научились работать с GIT через SourceTree, надеюсь этот инструмент вызвал у вас интерес.

OneScript & Precommit1C

Продолжаем формировать окружение разработчика. У нас есть GIT с которым мы можем работать с помощью SourceTree. Давайте попробуем поиспользовать GIT для 1C. Создадим новый репозиторий, в этот раз не будем указывать данные пользователя, посмотрим как поведет себя GIT в таком случае. Откроем конфигуратор → создадим обработку → сохраним ее в новый репозиторий. Закоммитим данное событие. Я буду работать с закладкой “Журнал/история” на мой взгляд она более наглядная. Индексируем → Добавляем описание → Пытаемся закоммитить → Получаем сообщение в котором GIT просит нас ввести данные пользователя, и это здорово, т.к. благодаря этой проверке все коммиты будут иметь автора и возможность связаться с ним по электронной почте. Что ж

давайте укажем имя и адрес электронной почты пользователя. Пока ещё мы можем себе позволить вводить не существующие данные. Снова попробуем закоммитить - успешно. Сделаем ещё коммит и проверим что мы можем перемещаться между коммитами. Например, добавим форму и в модуле формы добавим процедуру. Сохраним обработку и перейдем sourcetree. Снова закоммитим. Попереключаемся между коммитами - да все работает. Теперь добавим процедуру в модуль объекта обработки, сохраним ее и перейдем с sourcetree. Встанем на строку с незакомиченными изменениями и увидим что информации о том что именно изменилось в файле обработки отсутствует, мы просто видим что обработка в принципе изменился. Может это из-за того что мы ещё не выполнили коммит? Давайте создадим в этом же репозитории текстовый файл с каким-либо текстом и проверим так ли это. Нет, содержание файла ещё до коммита можно посмотреть. Но давайте выполним коммит сделанных с обработкой изменений и посмотрим - может тогда мы увидим изменения модулей. Коммит сделан, а содержимое модулей обработки мы не видим. Дело в том что сейчас наша обработка представляет из себя бинарный файл, а чтобы увидеть изменения модулей нам нужно разложить бинарный файл обработки на исходные, грубо говоря, текстовые файлы и в этом нам помогут два инструмента, это oscript и precommit1C, а точнее та его версия которая использует oscript.

Зайдем в браузер и в строке поиска наберем

oscript

и выполним поиск. Перейдем по ссылке <http://oscript.io/>. С самим инструментом мы поработаем позже, для решения текущей задачи нам нужно его скачать, установить и проверить. Жмем на кнопку “Скачать” и выбираем пункт windows instaler exe. Скачивание завершилось, запускаем установку

Шаг	Описание	Действие
1	Можем изменить каталог установки	Оставим без изменения и перейдем на следующий шаг
2	Можем изменить состав компонент для установки	Оставим без изменения и перейдем на следующий шаг
3, 4	Финальные	Жмем кнопки install и finish

После установки oscript должен стать доступным из консоли, давайте это проверим. Я перейду на диск C и в свободном месте нажму shift и кликну правой кнопкой мыши, в появившемся контекстном меню выберу пункт “Открыть окно команд” чтобы открылась консоль. В открывшейся консоли я напишу

where oscript

и нажму Enter. Получаю сообщение что не удалось найти файлы по заданным шаблонам. Может быть я что-то не так написал? Давайте для проверки в консоли выполним команду

```
where git
```

с Gitом все в порядке, мы видим путь значит команду набираем верно. Может oscript не прописался в системную переменную Path? Проверим. Нет он на месте. Дело в том что на некоторых машинах требуется перезагрузка чтобы обновилась среда, давайте выполним перезагрузку. И уже после перезагрузки зайдём в консоль и выполним команду

```
where oscript
```

видим путь, значит все в порядке oscript настроен.

Теперь нам нужно добыть oscript'овую версию precommit1C. Для этого нам потребуется аккаунт на github. К слову сказать "github" - это как и bitbucket, веб-сервис для хостинга репозиторий и командной разработки. Перейдем в браузер и в строке поиска напомним

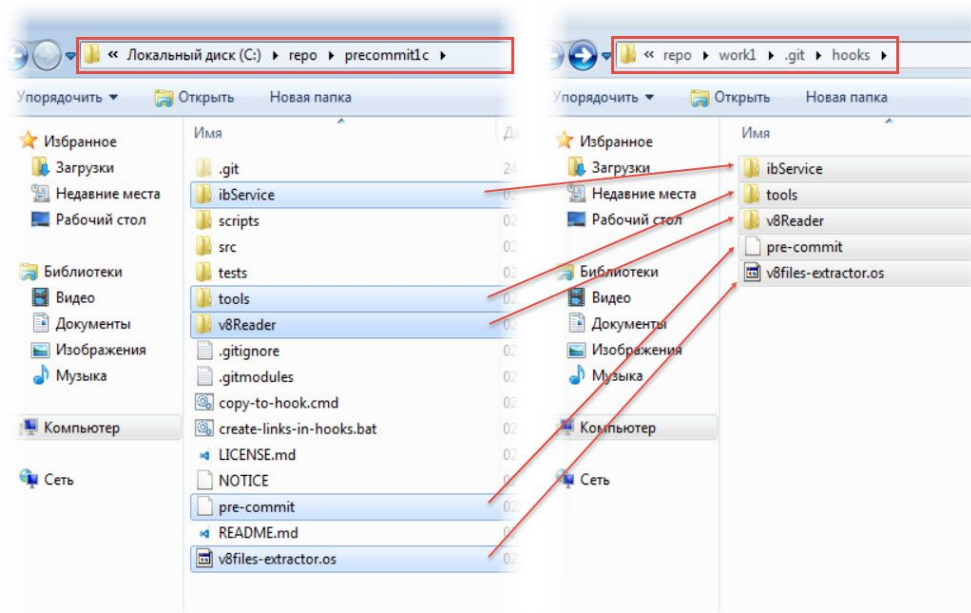
```
github
```

и выполним поиск. Перейдем по ссылке <https://github.com/> нажмем кнопку sign up чтобы зарегистрироваться. И вот тут я настоятельно рекомендую определиться с данными аккаунта, т.к. github своего рода социальная сеть, то здесь важны репутация и узнаваемость, это своего рода ваше портфолио. Итак, укажем имя пользователя, электронный адрес, пароль и нажмем кнопку "creat an assaunt". На следующем шаге нам нужно выбрать какими будут наши репозитории: частными или публичными, мы оставим публичный вариант, т.к. он бесплатный и нажмем на кнопку "finish sign up". Теперь когда у нас есть аккаунт на github выполним поиск инструмента, Для этого в строке поиска напомним

```
precommit1C
```

и выполним поиск. Первым в списке результат поиска мы видим precommit1C за авторством разработчика с ником "rumbaEO" в реальности это Евгений Сосна. Но на этом наш поиск не заканчивается, т.к. эта версия precommit1C использует интерпретатор python, а нам нужна версия precommit1C использующая интерпретатор oscript, найти ее мы сможем в форках этого продукта. В правом верхнем углу найдем кнопку Fork и кликнем по цифре справа от нее. Мы перешли в список всех форков, т.е. ответвлений этого продукта. Это значит что перед нами список разработчиков или сообществ разработчиков которые сочли полезным этот инструмент и взяли его себе для доработки под свои цели. Сейчас нас интересует форк сообщества xDrivenDevelopment, перейдем

на него, т.е. по ссылке <https://github.com/xDrivenDevelopment/precommit1c>. Видим что эта версия precommit1C является ответвлением и принадлежит сообществу xDrivenDevelopment. Справедливости ради отмечу, что автором этой версии precommit1C является Никита Грызлов. Перейдем на закладку “code” и в списке веток выберем ветку master и нажмем на кнопку clone or download. В открывшемся окне нажмем на кнопку “copy to clipboard”, чтобы скопировать в буфер обмена URL адрес этого репозитория для клонирования и помощью GIT. Вернемся в sourcetree и нажмем на кнопку “Клонировать/создать”. В открывшемся окне на закладке “Клонировать репозиторий” вставим из буфера обмена в поле “Исходный путь” URL адрес репозиторий, перейдем в поле “Целевой путь”, оно заполнилось автоматически, нам осталось указать путь к репозиторию из нашего каталога репозитория и нажать кнопку “Клонировать”. Пока выполняется клонирование мы можем установить флаг “Показывать полный вывод” чтобы видеть какие команды sourcetree отправляются в GIT. Итак, precommit1C склонировался и для работы с управляемыми формами он готов, а вот чтобы использовать его для обычных форм нужно: зайти в репозиторий precommita → перейти в каталог tools → скопировать файл v8unpack.exe → вставить его в каталог bin скрипта. Готово. Теперь важно понять что для каждого репозитория (созданного, подключенного или клонированного) нужно один раз но все таки настроить precommit. Шпаргалка есть в файле [readme.md](#) ну а мы в нашем репозитории в системном каталоге .git создадим каталог hooks. Затем скопируем следующие файлы и каталоги из репозитория precommit1C в каталог hooks нашего репозитория



Осталось заключительное действие. В нашем репозитории открываем консоль и выполняем команду

```
git config --local core.quotePath false
```


эта настройка нужна для того чтобы GIT корректно обрабатывал кириллические наименования файлов. На этом настройки precommit1C для этого репозитория закончены, посмотрим какую пользу нам это приносит.

Вспоминаем, мы хотели чтобы внешние обработки и отчеты разбирались на исходные файлы чтобы мы могли видеть их содержание в истории коммитов. Давайте сделаем какие-либо изменения в модуле формы обработки, например: добавим процедуру

```
&НаКлиенте
Процедура ПриОткрытии(Отказ)
    //Вставить содержимое обработчика
КонецПроцедуры
```

Сохраним обработку и перейдем в sourcetree ожидая увидеть незакоммиченные изменения. Встанем на внешнюю обработку, но содержания изменения мы так и не видим. Дело в том что precommit1C срабатывает в момент коммита, давайте это проверим: проиндексируем обработку, добавим описание и закоммитим. Коммит завершился и давайте посмотрим на его результат. Да, в списке закоммиченных файлов кроме обработки мы видим ещё файлы, но нас интересуют файлы с расширением bsl - это расширение файлов модулей 1C. Ещё давайте посмотрим изменилось ли что-то в нашем репозитории. Да - появился каталог "src" в котором, в каталоге соответствующем имени обработки есть файлы на которые precommit'ом была разложена обработка. Текстовый файл на котором мы делали проверку нам уже не нужен в нашем репозитории, поэтому давайте его удалим: встав на него и вызовем контекстное меню и выберем пункт "Удалить". К слову сказать - если бы этот файл был закоммичен, ты мы могли бы вернуть его состояние к состоянию на момент последнего по времени коммита, выбрав пункт контекстного меню "Отменить". На этом настройка инструментов oscript и precommit1C закончена, в следующих уроках закрепим пройденный материал.

BDDEditor & Behavior

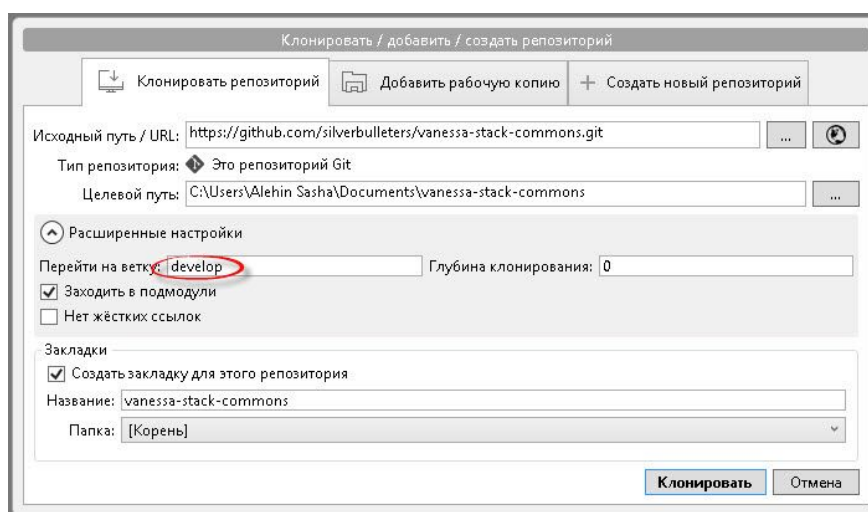
Продолжаем формировать окружение разработчика. Откроем браузер и перейдем на главную страницу github. Выполняя поиск на github обращайте внимание в каком контексте выполняется поиск. Если мы на главной странице, то поиск будет выполняться по всему github'у, но мы можем перейти в какой либо репозиторий и тогда поиск будет выполняться только по этому репозиторию. А если мы перейдем в какое-либо сообщество - то соответственно поиск будет выполняться в пределах сообщества. Итак, перейдем на главной странице, в строке поиска напомним

```
vanessa
```

и выполним поиск. Поясню почему ванесса. С одной стороны если произнести 1C на английский язык - получится "уон си", т.е. ванессы, а с другой стороны ванессы - это род бабочек в котором их бабочек множество видов, т.е. целое семейство. Вот и

получается что ванессы - это семейство продуктов для 1С, которые применяются при разработке по промышленным стандартам. А один конкретный продукт в таком семействе имеет префикс “vanessa”. В списке результатов поиска мы видим что один из продукты семейства vanessa принадлежат сообществу silverbulleters, и чтобы увидеть все продукты семейства vanessa мы должны перейти на страницу сообщества, для этого зайдём на страницу любого продукта с префиксом vanessa а из него на страницу сообщества silverbulleters, т.е. по ссылке <https://github.com/silverbulleters>. Сейчас нас интересуют два продукта: BDDEditor его автор Екатерина Золотарева и Behavior его автор Леонид Паутов, найдем и откроем их в отдельных вкладках, а так же добавим их себе в избранное.

Перейдем на страницу BDDEditor’a, скопируем его URL адрес в буфер обмена. Перейдем в sourcetree запустим клонирование внимание - ветки develop, т.е. разработческой ветки. Для этого укажем URL, нажмем Enter и раскроем расширенные настройки и пропишем



так же поступим и с продуктом behavior, пусть клонируется параллельно. Пока выкачиваются инструменты расскажу о них вкратце, т.к. позже мы ещё поработаем с ними.

Оба инструмента являются внешними обработками 1С:

- BDDEditor - используется для работы с обращениями заказчиков и создания на их основе feature файлов.
- Behavior - используется для того чтобы на основании feature файлов создать внешние обработки и выполнять сценарии, опять же на основании feature файлов.

Чтобы работать с BDDEditor его достаточно просто скачать, а вот для работы с Behavior потребуется интерпритатор Python и v8unpack, но об этом позже. Инструменты склонировались, запустим 1С и откроем BDDEditor. Выберем каталог в котором мы будем хранить наши feature файлы. Очень важно в каждом проекте/репозитории для хранения feature файлов использовать каталог features, именно на английском языке и именно с строчными буквами. Это связано с материалом из продвинутого курса, поэтому сейчас я не буду на этом останавливаться. Итак, создадим и выберем каталог

features

нажмем на кнопку “Добавить” и выберем пункт меню “Файл”. Дадим ему имя “Проверка генерации внешних обработок”. Видим что автоматически заполнилось поле редактирования, о том почему и зачем здесь написано то что написано поговорим позже, сейчас мы настраиваем инструмент Behavior. Отредактируем текст:

encoding: utf-8

language: ru

Функционал: Проверка генерации внешних обработок

Как участник базового курса разработки по промышленным стандартам

Хочу проверить работу инструмента vanessa-behavior

Чтобы использовать его в обучении и повседневной работе

Сценарий: Генерация epf

Допустим у меня есть feature файл

И у меня есть инструмент vanessa-behavior

Когда я в vanessa-behavior запускаю генерацию внешней обработки

Тогда рядом с feature файлом создается каталог а в нем внешняя обработка

и сохраним feature файл нажав соответствующую кнопку. Откроем Behavior и нажмем на кнопку “Загрузить” и выбираем пункт меню “Один файл”. Загрузка выполнена, перейдем на закладку Генерация EPF и нажмем кнопку “Создать, обновить шаблон обработки”. Получаем сообщение, что не установлен Python. Давайте решим эту проблему. Зайдем в браузер и в адресной строке напишем

python

перейдем по ссылке <https://www.python.org/> и нажмем на кнопку “Download”. Для операционной системы windows 7 я буду качать Python 2.7.1, если у вас ОС 7 хотя бы с первым пакетом обновлений - можете качать новую версию. Запускаем установку:

Шаг	Описание	Действие
1	Нужно выбрать вариант установки: только для себя или для всех пользователей	Оставим без изменения и перейдем на следующий шаг
2	Можем изменить состав каталог установки	Оставим без изменения и перейдем на следующий шаг
3	Можно выбрать состав компонент	В состав компонент для установки включим установку Python в path и перейдем на следующий шаг

4, 5	Финальные	Install, finish
------	-----------	-----------------

Перейдем в behavior и нажмем на кнопку создать. Снова видим сообщение о том что не установлен python. Проверим может он не прописался в path, выполним команду where python - все нормально, видим путь к каталогу. Дело в том что 1C запоминает системные переменные в момент открытия, поэтому давайте закроем все экземпляры 1C и откроем нужную конфигурацию и в ней откроем behavior, feature загрузилась автоматически, попробуем сгенерировать обработку. Обработка создана, значит behavior - настроен и мы можем переходить к следующему уроку.

Внимательный слушатель мог бы возразить, ведь говоря о настройке behavior кроме интерпретатора python я говорил об утилите v8unpack да, если бы мы ранее не скачали и не установили бы oscript, а затем не клонировали precommit1C и не скопировали из его каталога tools в каталог oscript'a bin файл v8unpack - behavior не смог бы сейчас создать файл внешней обработки 1C.

Allure & IrfanView & VSC

Заканчиваем формировать окружение разработчика. Откроем браузер и перейдем на главную страницу github, нам нужно перейти на страницу продукта vanessa-behavior. Т.к. я добавил этот продукт себе в избранное, то могу поступить следующим образом: кликнуть на свой аватар и выбрать пункт "your stars" и кликнуть по нужному репозиторию. Сейчас я хочу посмотреть его описание и интересоваться меня будет пункт "утилита для формирования отчетов о проверке". Эта созданная и поддерживаемая компанией Яндекс java-приложение, которое не требует установки и формирует достаточно наглядный отчет о выполнении сценариев на основании тех данные которые для нее сгенерирует behavior, чтобы посмотреть как это выглядит на практике выберем пункт "release notes" и "downloads zip". Рядом с каталогом геро создадим каталог "utils" - в нем мы будем хранить приложения не требующие установки. Разархивируем архив в каталог utils в соответствующий каталог. Перейдем в allure-commandline далее в каталог bin и скопируем путь, этот путь нам нужно добавить в системную переменную Path. Для этого зайдём в свойства компьютера → дополнительные параметры системы → переменные среды → системные переменные → Path → кнопка "Изменить" → добавим точку с запятой, если ее не было и вставим ранее скопированный путь → Ок. Откроем консоль и с помощью команды

```
where allure
```

проверим доступность. Видим путь, значит все ОК. Теперь чтобы этот отчет у нас формировался нужно установить jre. JRE - это минимальная реализация виртуальной машины для выполнения java-приложений. Зайдем в браузер, напомним поиск

перейдем по ссылке на <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

открывшейся странице подтвердим принятие лицензионного соглашения и скачаем соответствующий нашей операционной системе и ее разрядности установщик. Запустим установку и нажмем кнопку install. Готово, осталось только проверить работу отчета, для этого перейдем в behavior → загрузим ранее написанный feature файл → перейдем на закладку “Сервис” → установим флаг “Формировать отчет Allure” и укажем путь в который behavior будем помещать данные для формирования отчета выполним сценарий, проверим что в каталоге появился файл. Теперь, откроем из этого каталога консоль и выполним команду

```
allure generate .
```

после этого рядом с файлом должен появиться каталог, т.е. отчет сформирован и мы можем его посмотреть, для этого выполним команду

```
allure report open
```

Давайте я расскажу что произошло: allure поднял веб-сервер и отобразил результат формирования отчета во вкладке браузера. Закроем вкладку браузера и вернемся в консоль чтобы выйти из веб-сервера. Для этого в консоли выполним сочетание клавиш Ctrl+C и нажмем клавишу “y” и Enter.

Итак, отчет allure у нас есть, переходим к следующему инструменту. Для этого зайдём в браузер, напомним

IrfanView

и выполним поиск. Перейдем по ссылке <http://www.irfanview.com/> и выберем 32-х битный вариант. Скачаем и запустим установку:

Шаг	Описание	Действие
1	Настройка	Нажмем кнопку “uncheck all”, путь установки менять не будем и перейдем на следующий шаг
2	Сообщается о новой функциональности	Оставим без изменения и перейдем на следующий шаг
3	Предлагается ассоциировать приложение с какими-либо файлами	Оставим без изменения и перейдем на следующий шаг
4	Предлагается изменить расположения INI файла	Оставим без изменения и перейдем на следующий шаг

5	Финальный	Снимем оба флажка и нажмем на кнопку "Done"
---	-----------	---------------------------------------------

Проверять ifanview будем позже, а сейчас займемся ещё одним инструментом необходимым для завершения формирования окружения разработчика.

Зайдем в браузер, напишем

```
visual studio code
```

и выполним поиск. Перейдем по ссылке <https://code.visualstudio.com/> и в открывшемся окне нажмем кнопку "download". Запускаем установку:

Шаг	Описание	Действие
1	Приветствие	Перейдем на следующий шаг
2	Лицензионное соглашение	Принимаем и переходим на следующий шаг
3	Предлагается изменить каталог установки	Оставим без изменения и перейдем на следующий шаг
4	Предлагается изменить настройки ярлыков	Оставим без изменения и перейдем на следующий шаг
5	Дополнение контекстного меню	Установим флаги и перейдем на следующий шаг

Готово. В запущенном приложении VSC нажмем клавишу F1 и в появившемся поле напишем

```
ext inst
```

выберем пункт "Установить расширение" и напишем bsl - это официальное расширения файлов модулей 1C. Ожидаем появления кнопки "Перезагрузка" → перезагружаем. Готово, на этом формирование окружения разработчика - ЗАКОНЧЕНО!!!

Итоги

Теперь, когда мы так или иначе познакомились со всеми необходимыми инструментами, можно посмотреть на окружение разработчика в целом и подвести некоторые итоги. Итак, чтобы сформировать окружение разработчика на ОС windows мы первым делом установили платформу 1C:Предприятие. Далее, чтобы мы могли работать с системой контроля версий GIT мы установили .Net Framework 4.5, затем мы установили

сам GIT и чтобы удобней с ним работать - установили SourceTree. Затем, чтобы внешние отчеты и обработки разбирались на исходные файлы (во время коммита в GIT) мы научились подключать к репозиторию Precommit1C и выполнять настройку репозитория для корректной работы с кириллическими наименованиями файлов, а чтобы работал precommit1C мы установили OScript и поместили в его каталог утилиту v8unpack, что сократило нам время настройки vanessa-behavior и дало возможность разбирать на исходные файлы внешние обработки и отчеты с обычными формами. Позже мы ещё познакомимся с инструментом Oscript, ну а пока, для удобной работы с ним мы установили VSC и добавили к нему расширение BSL. Далее с github мы клонировали два продукта BDDEditor и Behavior, чтобы работал behavior мы скачали и прописали в системной переменной allure, напомним, чтобы работал allure - мы установили мини-виртуальную машину jre. Так же для работы behavior мы скачали и установили python и irfanview. В итоге получилось 14 продуктов и 2 аккаунта. На этом занятие по настройке окружения (в рамках базового курса) можно считать пройденным.

Домашнее задание

В рамках домашнего задания нужно:

- Отключить в GIT контроль выбранных файлов
- Свой внешний репозиторий на github с разложенными на исходники файлами
 - epf
 - erf
 - q1c
 - st

Не забывайте что внешний репозиторий, как и локальный то же нужно настраивать, т.е. указать имя и электронный адрес пользователя, а так же настройку для корректной обработки кириллических наименований файлов.

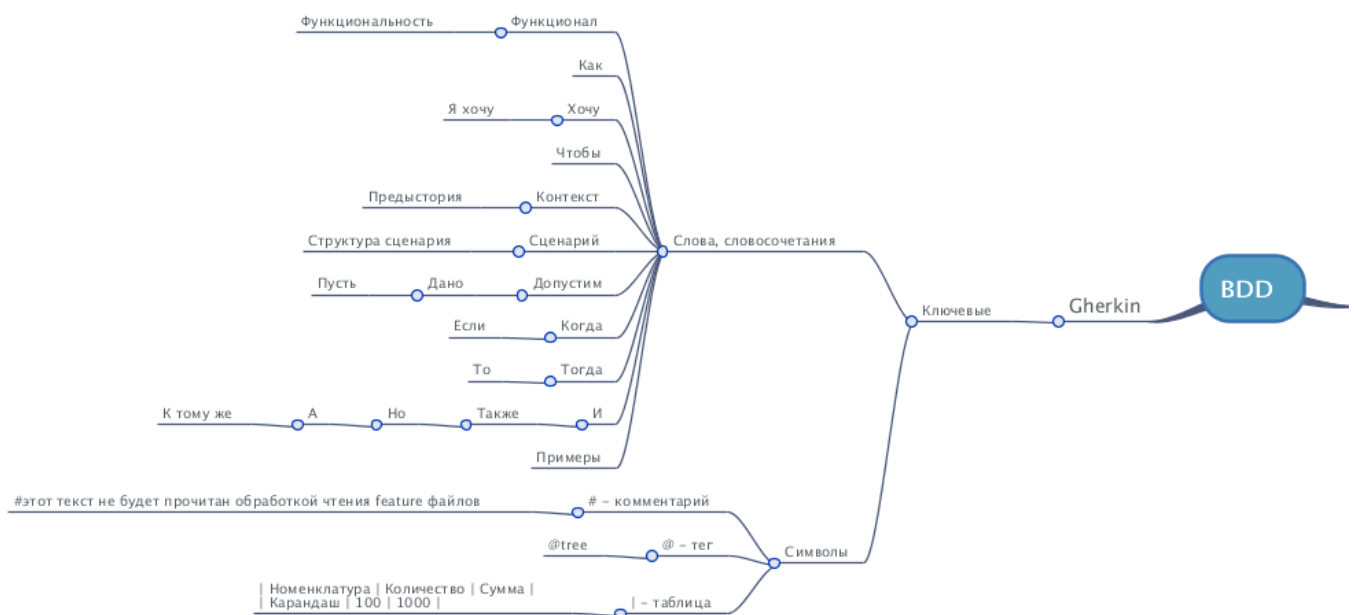
BDD

О теме

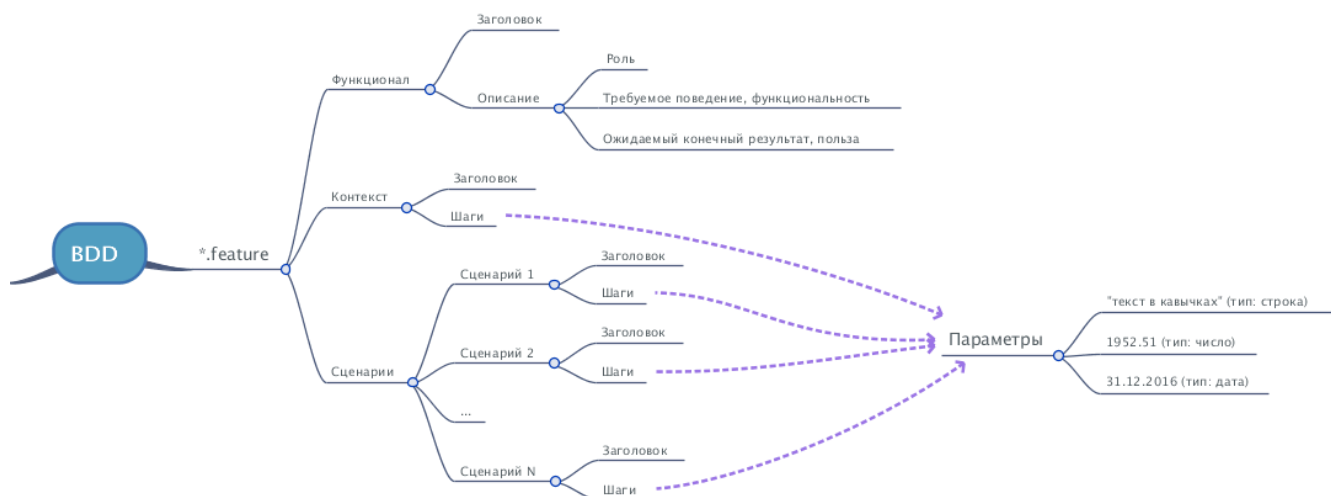
В промышленных стандартах разработки используются различные методологии и подходы. В этом курсе мы будем использовать методологию разработки через поведение или **Behavior Driven Development** сокращенно BDD. Чтобы направить ход ваших мыслей в нужное русло - разберем простой пример без углубления в предметную область. Например: мы как разработчики получили от заказчика письмо с темой “СРОЧНО” и таким содержанием: “сделайте мне сложение двух чисел”. Что ж давайте сделаем.

*.feature

По промышленным стандартам, основанием для разработки является только текстовый файл с расширением *.feature написанный на языке Gherkin (лучший вариант - когда этот файл написан вместе с заказчиком или аналитиком)



Итак, текстовый файл, коим является наш фича файл состоит из строк. В нем допускаются пустые строки, но не пустые строки - обязательно должны начинаться с ключевого слова или словосочетания или символа, т.е. предложение нужно писать в одну строку. В основном используются перечисленные на рисунке ключевые слова и словосочетания, но можно использовать и их аналоги. Как я и говорил ещё есть ключевые символы для комментирования, добавления тегов и создания таблиц. Сам же фича файл имеет определенную структуру:



Сначала идет функционал, его заголовок и описание в котором определяется роль, необходимое поведение и ожидаемый конечный результат. Далее идет контекст, но он не обязателен, т.е. в фича файле его может и не быть. Затем идут сценарии или структуры сценариев, т.е. в одном фича файле их сценариев и структур сценариев может быть несколько. Контекст, сценарии и структура сценариев состоят из шагов, а у шагов могут быть параметры, с типом: строка, число или дата.

Давайте преобразуем исходное требование в фича файл. Запустим 1С в режиме предприятия и откроем bddeditor. Создадим новый файл и дадим ему имя, чаще всего имя совпадает с заголовком функционала - это и было сделано автоматически. Заполним шаблон

```
# encoding: utf-8
# language: ru
```

Функционал: Сложение двух чисел

Как кассир

Хочу получать результат сложения двух чисел

Чтобы экономить на калькуляторах

Сценарий: сложение 10 и 15.5

Допустим у меня есть первое слагаемое равное 10

И у меня есть второе слагаемое равное 15.5

Тогда результат сложения двух чисел равен 25.5

Сохраним файл и закоммитим изменения с описанием “Преобразовал исходное требование в фичу”.

Обработка проверки поведения

Откроем Behavior и загрузим этот файл. Обратите внимание на колонку “Снипет” - в ней мы видим результат преобразования шага в имя процедуры, т.е. из шага было убрано ключевое слово, а слова в кавычках, числа и даты были перенесены в параметры процедуры. У каждого снипета должен быть адрес, т.е. путь к обработке в которой есть

процедура соответствующая снippetу. Давайте создадим такую обработку автоматически, перейдем на закладку “Генератор EPF” и нажмем на кнопку “Создать и обновить шаблоны обработок”. Перезагрузим сценарий, ожидая увидеть адрес снippetа в колонке “Обработка проверки”. Да - колонка заполнилась, это произошло благодаря тому что behavior рядом с фича файлом создал каталог “step_definitions” а в нем внешнюю обработку 1C. Закоммитим изменения “Создал обработку проверки поведения”.

Состояние шага

Теперь, когда у нас есть фича и есть соответствующая этой фиче обработка проверки поведения, мы можем запускать фичу на выполнение и результат выполнения наблюдать в дереве фич. Запускаем выполнение фичи и видим что процесс выполнения прервался. Так происходит когда шаг не реализован или когда шаг упал. Вообще, у шага есть три состояние:

Состояние	Цвет
Ожидает реализации (не реализован)	Желтый
Пройден	Зеленый
Не пройден (упал)	Красный

Давайте разберемся с каждым из них. Шаг будет считаться не реализованным если:

- После загрузки для него не определился адрес снippetа. Давайте посмотрим: удалим каталог “step_definitions”, перезагрузим фича файл и запустим выполнение. Видим что шаг подсвечивается желтым цветом.
- Адрес снippetа определился но в сообщении об исключении есть текст “Не реализовано”. Давайте откроем обработку проверки поведения в конфигураторе (если вы ее удалили на прошлом примере, то сгенерируйте ее снова) и перейдем в модуль формы. Здесь мы видим процедуры соответствующие шагам сценария. В теле этих процедур автоматически прописан метод вызова исключения с текстом “Не реализовано”.

```
&НаКлиенте
//Допустим у меня есть первое слагаемое равное 10
//@ДопустимУМеняЕстьПервоеСлагаемоеРавное(Парам01)
Процедура ДопустимУМеняЕстьПервоеСлагаемоеРавное(ПервоеСлагаемое) Экспорт
    ВызватьИсключение "Не реализовано";
КонецПроцедуры
```

И тут мы подходим к важному моменту: если во время выполнения процедуры соответствующей шагу произойдет исключительная ситуация и в тексте сообщения не будет “Не реализовано” - шаг будет считаться упавшим . Давайте изменим текст и запустим сценарий на выполнение. Да, шаг - красный и выполнение - прервано. Теперь

только когда процедура соответствующая шагу - отработает, шаг будет считаться пройденным. Давайте уберем вызов исключения и запустим фичу. Готово - шаг зеленый.

Программирование шагов

Теперь когда у нас есть фича, обработка проверки поведения и есть понимание как определяется состояния шага, мы можем запрограммировать шаги. Строгого регламента определяющего последовательность разработки функционала или программирования шагов - нет и чаще всего процессы идет параллельно. В нашем случае начнем с функционала. Требований к интерфейсу у нас пока нет, и не известно откуда будет использоваться функционал из какой-нибудь обработки или из документа, а может и из обработки и из документа. Я реализовал задачу следующим образом: создал клиентский общий модуль, в нем создал экспортную функцию в которую передается слагаемые она их суммирует и возвращает результат. Затем я создал общую форму и разместил на ней два реквизита и кнопку. В процедуре-обработчике события нажатия на кнопку я вызываю функцию сложения и сообщаю результат. Так же я добавил эту форму на начальную страницу, чтобы она появлялась при открытии конфигурации. Давайте посмотрим как работает функционал. Все в порядке и теперь давайте запрограммируем шаги проверки. Первым выполниться шаг "Допустим у меня есть первое слагаемое равное 10", т.е. будет вызвана соответствующая шагу процедура, в нее будет передан параметр и если мы его не сохраним то к третьему шагу в котором мы будем вызывать функцию суммирования значение параметра будет потеряно. То же самое касается и второго шага. Когда behavior генерировал обработку проверки поведения, он наполнил модуль формы необходимым количеством кода, в котором есть две глобальные клиентские переменные с типом структура:

- Контекст - для хранения значений между шагами
- КонтекстСохраняемый - для хранения значений между сценариями

Вот переменную "Контекст" мы и будем использовать. Сохраним в нее параметр из первого шага и второго шага. В третьем шаге мы вызовем функцию и проверим результат с параметром если не равны - вызовем исключение. Готово - сценарий пройден! Закоммитим изменения "Функционал реализован и проверен".

А теперь давайте представим что у нас функция не складывает два числа, а обращается к данным регистров, например: расчета, делает сложные вычисления и возвращает результат. Мы совместно с заказчиком придумали сценарий проверки этого функционала: в качестве параметров передавали в функцию сотрудника Иванов и период расчета Январь и результат расчета соответствовал проверяемому. Но после помещения функционала в рабочую базу появилась ошибка, когда функцию вызвали с параметрами сотрудник Петров и период расчета Февраль. В таком случае мы поступаем следующим образом:

- добавляем к фиче сценарий в котором воспроизводится ситуация создавшая ошибку
- при выполнении этот сценарий должен падать пока мы не исправим функционал

Давайте создадим похожую ситуацию в нашем функционале. Например, во время отладки мы прописали в коде возвращаемое значение и сценарий успешно проходит. А

когда пользователь открыл форму и сложил два числа сумма его не удовлетворила. Добавим сценарий с теми параметрами которые указал пользователь. Сценарий падает, исправляем функционал - оба сценария проходят - функционал проверен. Закоммитим изменения “Доработал фичу”.

Allure

Теперь давайте представим что у нас накопилось какое-то приличное количество фич и мы хотим выполнить их все одну за одной чтобы проверить весь функционал. Но выполнение всех фич может занять несколько часов, как же быть? Мы можем в течении рабочего дня запускать только те фичи над функционалом которых работаем, а выполнение всех фич запускать перед уходом домой. Скажу больше - у нас даже есть инструмент а в behavior заложен соответствующий функционал для формирования отчета о результате прогона всех фич. Для этого нам нужно на закладке “Сервис”:

- установить флаг “Формировать данные для отчета Allure” и указать путь к каталогу в который behavior будет сохранять эти данные
- установить флаг “Скриншот ошибок” и указать путь к каталогу в котором будут создаваться скриншоты.

Загрузим все фичи из каталога и запустим выполнение сценариев. Видим сообщение в котором говорится что данные для allure созданы. В каталоге куда behavior сохранил данные запустим консоль и выполним команду:

```
allure generate .
```

и когда она отработает выполним ещё одну команду:

```
allure report open
```

Готово.

Сейчас может быть скриншот ошибки избыточен, но позже когда мы научимся воспроизводить действия пользователя - он будет к месту и поможет сэкономить достаточно много времени и сил. Т.к. отчет allure и скриншот ошибок в течении рабочего дня нам не нужны - снимем соответствующие флажки и потушим виртуальный сервер выполнив соответствующие команды в консоли:

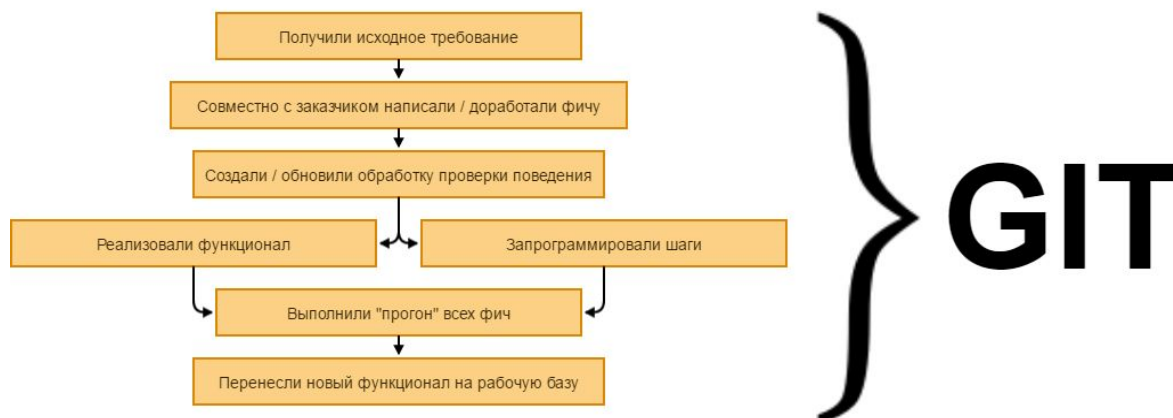
```
Ctrl + C
```

```
y
```

```
Enter
```

Итоги

Давайте подведем итоги темы разработки через поведение. Теперь мы знаем что требование заказчика - это ещё не основание для разработки, а срочность и тяжесть складывающихся порой ситуаций не повод менять методологии которые работают на методологии которые не работают и “в долгую” приносят вред. Схематично работу по BDD можно представить так



Домашнее задание

Обработать по BDD такое требование “Мне нужно чтобы 1С мне показывала возраст человека в месяцах”.

Организация процесса

О теме

В этой теме речь пойдет о работе с заказчиками. Мы посмотрим как устроен процесс работы с заказчиками в подавляющем большинстве организаций России и чем этот процесс плох, а потом покажу как он должен выглядеть хотя бы по итогам базового курса разработки по промышленным стандартам.

“Лоскутная” автоматизация

Давайте посмотрим как выглядит средне-статистический процесс работы с заказчиком.

Итак, есть заказчик который работает с конфигурацией базы данных, и есть разработчик который работает с основной конфигурацией. Возможно конфигурации подключены к хранилищу, а возможно - нет. По мере необходимости заказчик, как умеет формулирует пожелания к конфигурации своими словами либо пишет, как он считает техническое задание на одном листе формата А4. Разработчик как-то понимает эти пожелания, может быть что-то уточняет и как-то реализует, как-то проверяет (самостоятельно либо вместе с заказчиком) и когда “все готово” → переносит новый функционал на конфигурацию базы данных и пишет инструкцию для заказчика.

Заказчик начинает работать с новым функционалом и тут возможны следующие варианты развития событий:

- вполне возможно, что функционал на 100% устраивает заказчика и разработчик может заняться другой задачей
- а может быть и так что кто-то не так понял либо кто-то не так объяснил и когда заканчиваются споры разработчик занимается доработками нового функционала
- и уж совсем неприятный но к сожалению не редкий случай, когда новый то функционал работает - а старый и проверенный функционал - не работает либо работает не так как раньше.

Во втором и третьем случае разработчику вместо новой задачи приходится заниматься исправление ранее работавшего функционала без каких-либо гарантий что ещё какой-нибудь функционал перестанет работать. Все это отрицательно сказывается на репутации разработчика и на деловых отношениях с заказчиком. Возможно в организации есть:

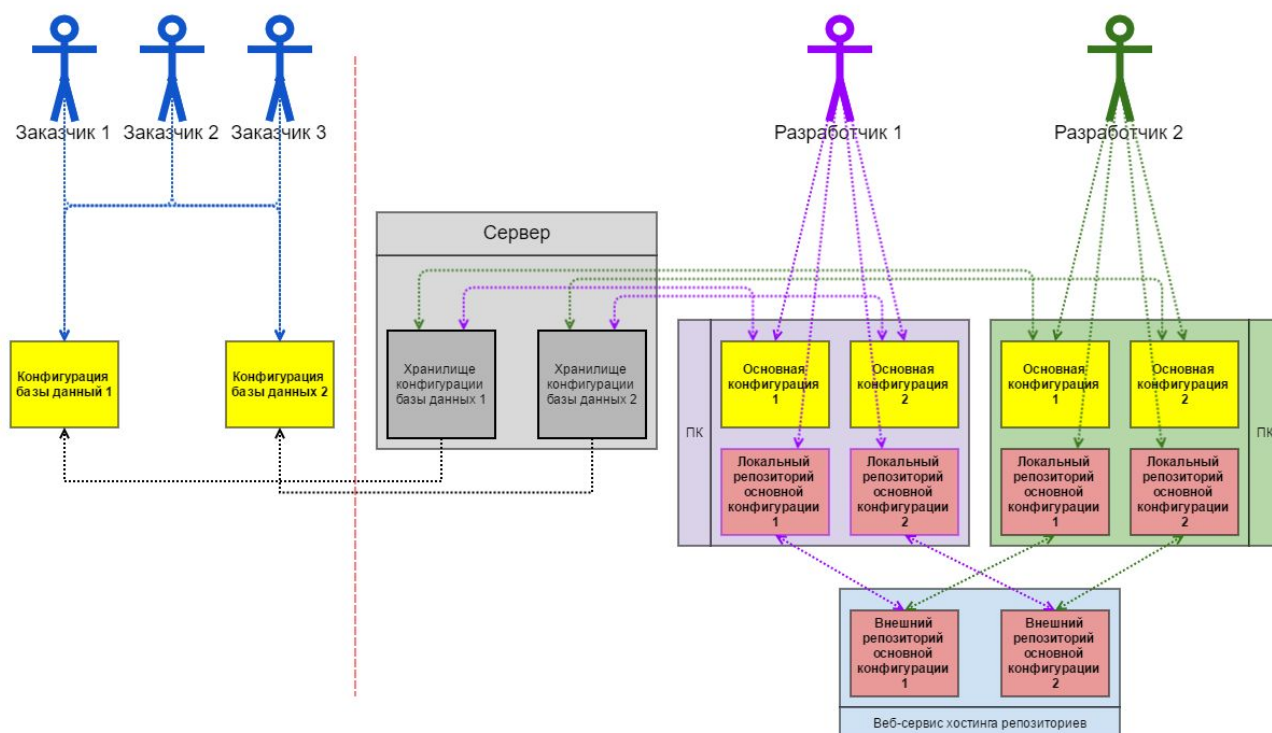
- аналитик, который видит картину в целом и пытается подружить заказчика с разработчиком
- тестировщик, который испытывает новый функционал и в случае ошибок возвращает его на доработку разработчику
- “документатор” (простите за выражение □), который общается с разработчиком и пишет красивые инструкции для заказчика, чтобы разработчик на это не отвлекался

Чаще всего на таких отдельно выделенных сотрудниках экономят, но даже если бы они и были, то результат их труда:

- зависел от их опыта и навыка доступно объяснять и правильно понимать
- устарел либо не соответствовал текущей функциональности
- хранился бы либо в чей-то голове либо в электронных переписках или томах документации на чтение и осознание которых может уйти не один день а решение нужно уже сейчас.

Промышленные стандарты

Теперь давайте посмотрим как в рамках базового курса выглядит процесс разработки по промышленным стандартам (предвосхищая ваш вопрос: да, в рамках продвинутого курса процесс гораздо шире). Усложним задачу:



Есть несколько заказчиков которые работают с несколькими конфигурациями базы данных и есть несколько разработчиков который работает с несколькими основными конфигурациями. Конфигурации, обязательно подключены к хранилищу. Кроме конфигуратора у разработчика настроено окружение и есть:

- локальный репозиторий
 - соответствующий основной конфигурации
 - с настроенным `precommit1C`
 - и в котором есть каталог features (а лучше все каталоги по [bootstrap](#))
- есть внешние репозитории ОБЩИЕ для всех разработчиков и соответствующие основным конфигурациям

Что же происходит когда заказчик понимает что в какой-то из конфигураций нужно что-то изменить? Да, заказчик может:

- написать разработчику письмо
- составить тех. задание
- договориться о встрече чтобы своими словами объяснить ситуацию

но итог будет один: *исходные требования будут зафиксированы в системе контроля версий, в соответствующем конфигурации репозитории, а разработчик возьмет это требование в работу только тогда, когда совместно с заказчиком, а лучше но не обязательно: совместно с заказчиком и аналитиком будет написан фича файл.*

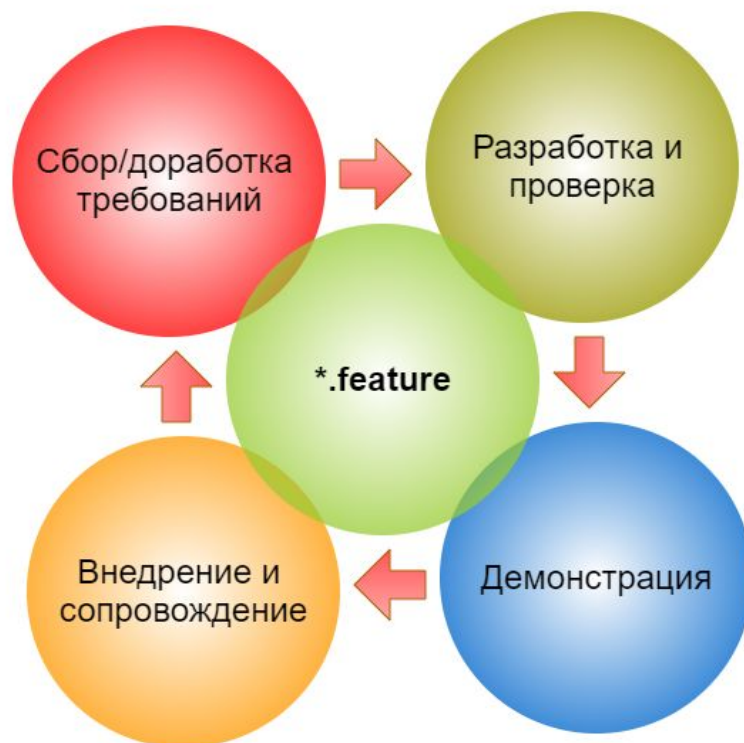
- В фича файле разработчик совместно с заказчиком поймет и зафиксирует роль того сотрудника который будет использовать функционал, т.е. банально - роль того кто будет нажимать кнопки.
- В фича файле разработчик совместно с заказчиком поймет и зафиксирует краткое определение функционала, так сказать суть задачи.
- в фича файле разработчик совместно с заказчиком поймет и зафиксирует какой бизнес-эффект, какая осязаемая польза будет от этого функционала.
- В фича файле разработчик совместно с заказчиком поймет и зафиксирует сценарии проверки работы функционала, благо Gherkin и создавался чтобы быть понятным пользователю и полезны разработчику.

Т.к. фича файл лежит в репозитории, все его изменения сохраняются, разработчик совместно с заказчиком могут изменять описание функционала

- корректировать
 - роль
 - краткое описание
 - ожидаемый конечный результат
- добавлять, изменять, удалять сценарии и шаги

и все эти изменения сохраняться в истории, так что будет видно какая строчка на какую поменялась и зачем. Так мы работаем на этапе сбора/доработки требований. Далее - разработка через поведение, т.е. разрабатываем проверяя. Разработчик создает обработку проверки поведения и программирует шаги чтобы те стали зелеными а сценарии в целом - пройденными. Но перед демонстрацией функционала заказчику, он получает себе из внешнего репозитория все фичи его коллег и выполняет прогон всех фич, т.е. и тех которые возможно он не писал. В итоге прогона всех фич разработчик получает представление как его функционал взаимодействует с остальным функционалом, т.е. уже на этом этапе может быть выявлена ошибка, а не на этапе использования. Когда все ошибки устранены и сценарии всех фич пройдены, можно переходить к следующему этапу - демонстрации функционала заказчику. По итогам этого этапа функционал либо отправляется на доработку, НО только после того как суть доработки будет отражена в фиче либо функционал будет перенесен на рабочую конфигурацию базы данных.

Итак, заключительный этап - сопровождение, т.е. снабжение пользователя необходимой документацией и если вы ещё не догадались - это фича и в behavior есть функционал для автоматического формирования инструкции с картинками. А финалисты курса - научиться использовать генератор “живой” пользовательской документации. Давайте посмотрим на процесс в целом:



Feature файл, он же спецификация на примерах, он же исполняемая спецификация - красной лентой проходит по всем этапам и гарантирует:

- работу функционала, в т.ч. и того что был реализован ранее, разумеется - если он покрыт сценариями
- гарантирует взаимное понимание функционала заказчиком и разработчиком
- а так же гарантирует актуальность документации

Как вам такой подход к процессу? И по сути, ничего такого уж нового и нет - все тот же 1С все тот же конфигуратор, только подход эффективней. Единственное что по настоящему может оказаться для вас новым и по началу сложным - это Gherkin, но и это вполне преодолимо. Давайте ещё немного поговорим о Gherkin а когда перейдем к практической части - мы разберемся с ним детальнее. Предпосылкой к появлению единого языка, коим является Gherkin стал тот факт, что заказчиков знающих и способных сформулировать что им действительно нужно - единицы, порой реализация функционала занимает меньше времени чем погружение в контекст задачи. Т.е. сама по себе предметная область может быть, и чаще - является сложной, а то что сотрудник разбирается в ней - не означает что он может сформулировать требования по доработке функционала связанного с этой предметной областью. Полагаю что не только на моей практике заказчики формулировали требования

- в виде нескольких предложение в устной форме
- в виде электронного письма со скриншотом и описанием задачи в теме письма
- в виде нескольких листов разноцветным текстом со скриншотами

Все эти формулировки были созданы разными людьми так как у них это получается. А теперь прикинем, сколько времени нужно затратить чтобы хотя бы осознать то о чем человек просит? До бесконечности. Если не связаться с этим человеком и не попросить

уточнений: что же он имел в виду. Выход из этой ситуации - единый стандарт, единый язык на котором разработчик совместно с заказчиком достигнут взаимного понимания предметной области и отразят модель предметной области в фича файлах. Чтобы разложить все по полочкам, давайте воспользуемся ассоциативной памятью, а именно вспомним как мы в школе решали задачи. У нас же был целый стандарт решения задач, некий типовой шаблон оформления, вспоминаете?

Дано: $AB = 5 \text{ км/ч}$	Решение: $x = 0.5 \text{ ч}$ $AB = 25.4 \text{ км/ч}$ $AB = 5 \text{ км/ч} \cdot 0.5 \text{ ч} = 2.5 \text{ км}$ $2.5 \text{ км} = 2.5 \text{ км}$
Найти: $x = ?$ $AB = ?$	$AB = 5 \text{ км/ч} \cdot 0.5 \text{ ч} = 2.5 \text{ км}$ Ответ: $AB = 2.5 \text{ км}$ $AB = 2.5 \text{ км}$

Смотрите сами: в шаблоне нужно что-то найти, а в фиче - реализовать функционал с тремя неизвестными

- Как X
- Хочу Y
- Чтобы Z

,где

- z - это роль
- y - требование или краткое описание функционала
- z - ожидаемый конечный результат

в шаблоне есть три секции: дано, решение, ответ и в фиче то же, но не обязательно, три секции: допустим, когда, тогда. С их помощью мы описываем сценарий, грубо говоря - пример. В секции “допустим” мы описываем исходную ситуацию. В секции “когда” мы описываем действия, некую последовательность событий. В секции “тогда” мы описываем результат, который мы хотим получить в этом сценарии. Остается учитывать что в фиче может быть несколько сценариев и единый контекст. А контекст и сценарии состоят из шагов. В итоге получаем текст, который поймет даже бабушка кассир, но в то же время позволит избежать не однозначного трактования.

Итоги

Итак, подведем итоги темы организации процесса. Полагаю мне удалось отразить в этой теме преимущество и эффективность работы по промышленным стандартам. Но цель у меня была несколько больше, а именно снабдить вас аргументами для продвижения и внедрения промышленных стандартов разработки в вашей организации, в вашей команде.

Домашнее задание

В рамках домашнего задания нужно написать скрипт для формирования отчета allure. В прошлой теме для формирования отчета allure мы в консоли выполняли команды вручную, давайте автоматизируем этот процесс. Из инструментов у вас все есть, а именно сам oscript и vsc с расширением bsl. Источником информации для вас будет [страничка](#) синтакс-помощник на сайте oscript'a. Начните с того что создайте файл с расширением *.os и откройте его в vsc.

Практика применения

О теме

В этой теме мы разберем примеры использования промышленных стандартов в частности разработки через поведение в тех случаях когда конфигурация находится на поддержке и когда в конфигурации включена возможность изменения. Напомню, для разработки через поведение на 1С мы используем два инструмента: `bddeditor` и `behavior`. Так вот `behavior` активно использует функционал автоматизированного тестирования 1С. Со знакомства с этим функционалом мы и начнем.

Автоматизированное тестирование

Функционал “автоматизированное тестирование” присутствует в платформе 1С начиная с версии 8.3 и доступен только для управляемых приложений. Чтобы с ним работать нужно запускать 1С в режиме предприятия с разными ключами. Начнем с ключа

`/uilogrecorder`

он дает возможность записывать действия пользователя и сохранять их в файл. Давайте запустим 1С с этим ключом, для этого через окно запуска перейдем в настройки подключения и добавим ключ. Запустим 1С в верхнем правом углу появились три кнопки для начала, прерывания и завершения записи. Посмотрим что будет если запустить и прервать запись. Как и ожидалось - ничего. Теперь запустим и завершим запись - наблюдаем следующее поведение: открылось окно создание текстового документа с текстом на языке xml. Давайте закроем его без сохранения и снова запустим запись, только в этот раз выполним поиск и только после этого завершим запись. Открылось окно и в нем есть текст, т.е. наши действия были зафиксированы и описаны на языке xml, этот файл принято называть “журнал записи действий пользователя”, но сам по себе журнал записи действий пользователя нам интересен тем что из него с помощью специальной внешней обработки можно получить код на встроенном языке 1С который будет воспроизводить записанные действия. Перейдем на сайт its.1c.ru (платная подписка не потребуется) и в строке поиска введем “преобразование журнала действий пользователя” и выполним поиск. Скопируем отработку и откроем в 1С предприятие, чтобы из журнала действий пользователя получить код воспроизведения действий пользователя нужно в обработке преобразования:

- установить флаг “Генерировать код подключения к тест клиенту”
- установить флаг “Разбивать результат на процедуры...”
- выбрать вариант “Преобразовать текст”
- скопировать журнал действий в соответствующее поле и нажать кнопку “Преобразовать”

В соответствующем окне видим код на встроенном языке 1С. Давай те перейдем в конфигуратор, создадим обработку и перенесем в нее полученный код. Давай те проанализируем код. Создается новый объект ТестируемоеПриложение и присваивается переменной ТестовоеПриложение. Запускается цикл продолжительностью 60 секунд, в котором выполняются попытки установить соединение с клиентом тестирования. Если соединение не установится то выполнение процедуры будет прервано. Если же удастся установить соединение то будут выполнены процедуры идущие после цикла. Итак, получаем подчиненные окна тестируемого приложения и определяем основное. В основном окне находим кнопку поиска, с помощью метода НайтиОбъект. И нажимаем на нее с помощью метода Нажать. Далее находим окно поиска и в нем находим поле и заполняем его. Нажимаем кнопку поиска. Сохраним обработку, готово. Теперь у нас есть обработка, которая будет выполнять код воспроизведения действий пользователя. Хорошо, но тут возникают два вопроса

- где запускать эту обработку
- и где будут отображаться действия пользователя

Разобраться с этими вопросами нам помогут ещё два ключа для запуска 1С в режиме менеджера тестирования

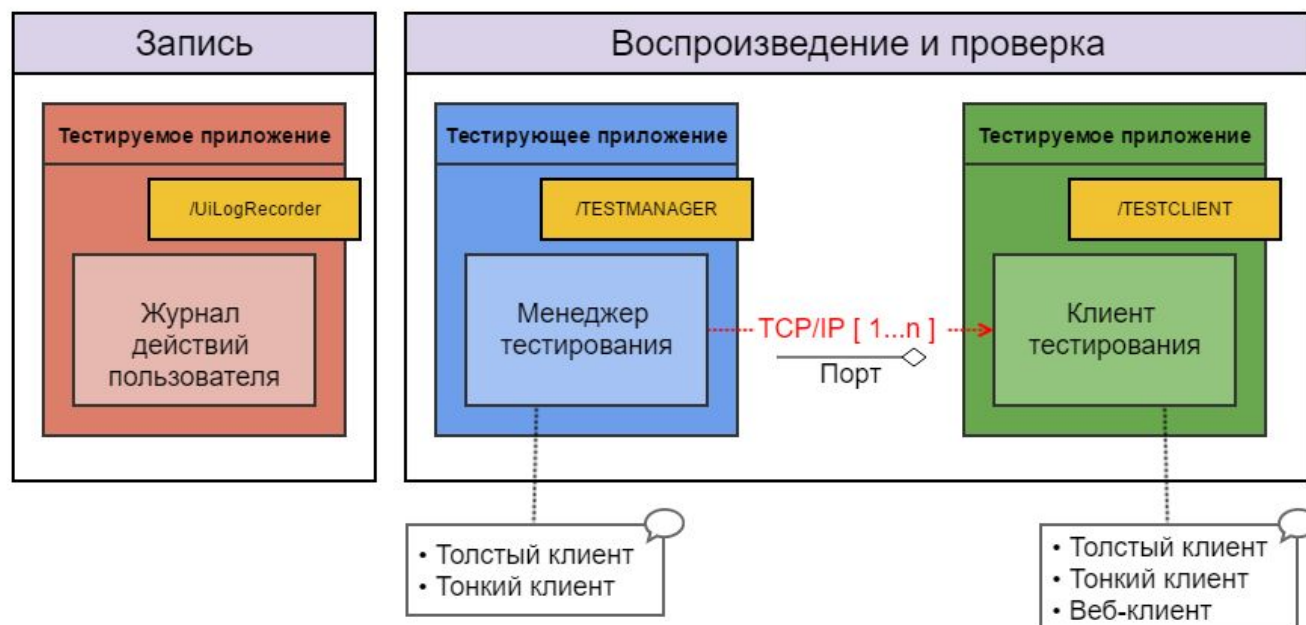
```
/testmanager
```

и режиме клиента тестирования

```
/testclient
```

как вы понимаете на момент исполнения кода воспроизведения действий пользователя должны быть запущены и менеджер и клиент тестирования. Давайте запустим менеджер тестирования: изменим параметр и клиент тестирования, так же изменим параметр. Откроем в менеджере тестирования нашу обработку, ожидаем что когда нажмем на кнопку то в клиенте тестирования будут воспроизведены действия пользователя, проверим. Видим на первый взгляд странную картину: похоже действия пользователя были воспроизведены в клиентском приложении запущенном с ключом записи действий пользователя. Давай те проверим, запустим обработку заново. Так и есть. Дело в том что ключ записи действий пользователя дает возможность не только записывать но и воспроизводить действия пользователя, грубо говоря - это тот же клиент тестирования но с кнопками для записи журнала действий пользователя. В том случае когда запущено несколько клиентов тестирования поиск и подключение будет выполнено в пределах порта к первому по времени запуску клиенту тестирования. Именно такая ситуация у нас и получилось, ведь запуская клиентов тестирования мы не указывали порты и был использован порт по умолчанию - 1538. Давайте на этом примере разберем как работать с несколькими клиентами тестирования. Укажем порт в запуске клиента тестирования и в подключении, например - 1539. Воспроизведем ситуацию и проверим что действия пользователя воспроизводятся на клиенте тестирования. Готово.

Теперь, когда мы пусть даже на простом примере но все же познакомились с функционалом автоматизированного тестирования предлагаю обратиться к теории и разложить все по полочкам.



Итак, функционал автоматизированного тестирования дает нам возможность записывать, воспроизводить и проверить результат действий пользователя. Для записи используем режим записи журнала действий пользователя, для воспроизведения и проверки используем менеджер тестирования и клиент тестирования, а так же обработку воспроизведения действий пользователя код которой получен из журнала действий пользователя. Менеджер устанавливает соединение и передает команды на сторону клиента

клиент их воспроизводит и полученные данные по запросу менеджера передает на сторону менеджера для анализа.

Чтобы перейти к практике знаний о автоматизированном тестировании у нас достаточно, но если вы хотите поглубже вникнуть в данный функционал, то на сайте its есть [статья](#) в которой можно прочитать и скачать обработку с примером. Так же в синтакс-помощнике в ветке Общие объекты → Автоматизированное тестирование есть описание функционала.

Фичи из “воздуха”

Перед тем как мы перейдем к практической части, давайте все те же действия пользователя воспроизведем с помощью инструмента *vanessa-behavior*. Для этого запустим менеджер тестирования и откроем в нем *behavior*. Перейдем на закладку “Работа с UI” и нажмем кнопку “Начать запись поведения”. После нажатия на эту кнопку должен открыться клиент тестирования. Он открылся, но программно была запущена запись действий пользователя, о чем подсказывает сообщение. Итак, давайте откроем окно поиск и что-нибудь поищем. Теперь вернемся в менеджер тестирования и завершим запись действий пользователя. Что же мы видим? Появился текст фичи и в нем есть

шаги, как это произошло? Этот функционал носит название “фичи из “воздуха”. Он, как и обработка преобразования журнала действий пользователя, анализирует текст xml полученный в результате записи действий пользователя и на его основании формирует только не код воспроизведения а шаги сценария. Больше того для каждого из этих шагов уже созданы и запрограммированы процедуры проверки шагов, они находятся в библиотеке (путь к ней заполняется автоматически при открытии инструмента). Скопируем текст фичи, перейдем в bddeditor и создадим новый файл. Скопируем в него текст и сохраним. Откроем новую фичу в behavior. Обратите внимание на серые строки - это значит что процедуры проверки шагов, они же снипеты, были найдены не в соответствующей фиче обработке проверки поведения, а в соседних обработках из того же каталога что и сама фича или behavior. Разместим рядом менеджер и клиент тестирования и запустим выполнение сценария. Как вы видите, действия пользователя успешно воспроизведены и мы при этом не написали ни строчки кода, собственно от сюда и название функционала “фичи из “воздуха”.

Конфигурация на поддержке или базовая

Давайте представим такую ситуацию: у нас на обслуживании бухгалтерия предприятия проф 3.0. Конфигурация полностью на поддержке без возможности изменения, т.е. все доработки нужно делать внешними обработками и отчетами. И вот мы получили письмо от заказчика. Из письма мы понимаем что нужно загружать материалы и есть пример файла из которого нужно эти материалы загружать. И ещё представим что мы только начинаем работать с этой или в этой организации и это наша первая задача. Но мы времени зря не теряли и уже успели:

- настроить окружение разработчика
- а так же добыть и поднять копию рабочей инфобазы (в моем случае это демо-база)

что ещё нам нужно? вспоминаем:

- нужен локальный репозиторий соответствующий конфигурации - т.е. мне как сотруднику для разработки
- а так же нужен внешний (желательно приватный) репозиторий - чтобы обмениваться с коллегами, ну и как дополнительное место хранения наработок, а т.к. денег у организации на систему учета задач - нет, то запросы пользователя будем учитывать в issues.

В рамках курса хранить репозитории и вести задачи я буду на bitbucket, благо на Bitbucket можно создавать приватные репозитории бесплатно. Но вы должны понимать что такие ресурсы как github и bitbucket просто используют технологию git и с таким же успехом подобный git-сервер вы можете организовать у себя, своими силами, но это уже тема для продвинутого курса.

Итак, создадим внешний репозиторий, чтобы обмениваться фичами с коллегами и выполнять прогоны всех фич, т.е. проверку всего функционала. Склонируем внешний репозиторий себе на рабочий ПК и настроим его:

- precommit1C
- кириллические наименования
- каталог features

Готово!

Теперь зафиксируем требование в issues, чтобы у нас появилась задача к которой мы могли бы привязывать все наши коммиты. Готово!

Что у нас дальше по процессу? начинаем разработку? НЕТ, совместно с заказчиком мы пишем фичу. Принципиального значения не имеет кто является инициатором задачи, в роли мы указываем того кто будет использовать функционал. В нашем примере инициатором и пользователем будет бухгалтер. Открываем bddeditor и пишем:

- название функционала: *Загрузка материалов*
- роль: *Бухгалтер*

хорошо, продолжаем предложение. Как бухгалтер я хочу. А чего в рамках данной задачи хочет бухгалтер? Пишем:

- хочу: *обработку для автоматической загрузки материалов из файла*

хорошо, а какая польза от реализации этого функционала? Пишем:

- чтобы: *экономить время и избежать ошибок заполнения*

хорошо, пользовательская история - готова!

Как бухгалтер

Хочу обработку автоматической загрузки материалов из файла

Чтобы экономить время и исключить опечатки

Сохраним текущий результат и перейдем к сценариям. Заголовки или названия сценариев должны обозначать свое отличие от других сценариев. Этот сценарий назовем "загрузка материалов из файла materials.json". Когда мы пишем фичу, мы помним что она должны быть человекочитаема. Желательно чтобы в сценарии было три секции:

- исходная ситуации (ключевое слово "Допустим") - в этой секции мы просто проверяем какие то условия
- затем секция действия (ключевое слово "Когда") - в этой секции мы описываем действия пользователя (например: когда я выбираю..., и я нажимаю, и я ввожу... и т.д.)
- и наконец секция проверки действия (ключевое слово "Тогда") - в этой секции проверяется результат действий пользователя с условиями описанными в секции "Допустим".

Нужно стремиться к тому чтобы в сценарии было три секции, но бывает что вполне читаемые и рабочие фичи получаются из одной или двух секций. Давайте попробуем описать эти три секции, с учетом того что нам нужно проверить успешность загрузки материалов. А как это проверить? Ответ очевиден, до загрузки - не было материалов, после загрузки - материалы появились. Опишем секцию допустим.

Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" нет элементов

теперь опишем действия бухгалтера (именно такая у нас в рамках этой фичи роль)

Когда я открываю обработку загрузки материалов
И я выбираю файл
И я выбираю группу с наименованием "Материалы"
И я нажимаю кнопку загрузить

осталось проверить результат

Тогда материалы загрузились корректно

Сохраняем фичу, но разрабатывать не спешим. Представьте что в файле придет тот материал который уже есть в базе, т.е. нужен ещё сценарий в котором проверим синхронизацию, грубо говоря нам нужно проверить что наша обработка загрузки не будет создавать дубли материалов. С бухгалтером условились что синхронизировать материалы будем по коду. Добавляем сценарий и заголовок у него будет такой “Загрузка материалов из файла materials.json с проверкой синхронизации”, а шаги будут такие

Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" есть элементы
Когда я открываю обработку загрузки материалов
И я выбираю файл
И я выбираю группу с наименованием "Материалы"
И я нажимаю кнопку загрузить
Тогда материалы загрузились корректно

Поясню зачем нам понадобилась группа материалы. Во-первых, это не противоречит логике задачи. Во-вторых, группа нужна для того чтобы отделить пользовательские данные, которые к нам пришли с копией рабочей инфобазы от данных с которыми мы будем работать в рамках фичи. Представьте что мы запрограммируем шаги и запустим сценарий на выполнение в первый раз и произойдет следующее:

- выполниться проверка того что материалов в группе нет
- воспроизведутся действия пользователя в результате которых загрузятся материалы
- выполниться проверка того что материалы в группе есть

а если мы запустим сценарий на выполнение ещё раз, то на первом же шаге, на проверке того что материалов в группе нет, мы получим исключение и шаг упадет, т.к. материалы остались после выполнения первого сценария. Так мы знакомимся со следующим правилом: *сценарии должны быть независимы друга от друга и должны готовить для себя окружения*. А в целом все фичи из каталога features должны успешно прогоняться не только на вашем рабочем ПК но и ПК вашего коллеги и на том ПК на котором будете демонстрировать функционал заказчику. Для этого, в нашем случае, нужно удалять элементы справочника Номенклатура а сделать это быстрее с отбором по группе.

Итак, у нас в фиче два сценария, а каждый сценарий должен готовить для себя окружение, некую исходную ситуацию. В первом сценарии элементов в группе материалы быть не должно, а во втором сценарии наоборот - должно быть два элемента да ещё такие которые есть в файле. Решить эту задачу нам поможет контекст. Добавим шаг

Допустим я подготавливаю группу с наименованием "Материалы" справочника "Номенклатура"

сохраним фичу, проверим что бухгалтеру данная фича ясна.

```
# encoding: utf-8
# language: ru
```

Функционал: Загрузка материалов

Как бухгалтер

Хочу обработку автоматической загрузки материалов из файла

Чтобы экономить время и исключить опечатки

Контекст:

Допустим я подготавливаю группу с наименованием "Материалы" справочника "Номенклатура"

И Я запускаю сценарий открытия TestClient или подключаю уже существующий

Сценарий: Загрузка материалов из файла materials.json

Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" нет элементов

Когда я открываю обработку загрузки материалов

И я выбираю файл

И я выбираю группу с наименованием "Материалы"

И я нажимаю кнопку загрузить

Тогда материалы загрузились корректно

Сценарий: Загрузка материалов из файла materials.json с проверкой синхронизации

Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" есть элементы

Когда я открываю обработку загрузки материалов

И я выбираю файл

И я выбираю группу с наименованием "Материалы"

И я нажимаю кнопку загрузить

Тогда материалы загрузились корректно

Закоммитим совместный результат и приступим к разработке. Загрузим в behavior нашу фичу, создадим обработку проверки поведения. Важный момент: мы сейчас создали обработку проверки поведения с теми шагами которые есть в фиче, но в ходе разработки нам может потребоваться скорректировать шаги и в таком случае мы можем по той же кнопке регенерировать обработку проверки поведения но нам потребуется переоткрыть обработку, а можем установить флаг "Выводить текст вместо регенерации" чтобы получить текст и скопировать нужные фрагменты кода, соответственно переоткрывать обработку в таком случае не нужно. Объясняя этот важный момент я обмолвился что мы можем корректировать фичу в ходе разработки, т.е. без участия заказчика. Возможно вы удивлены, но разработчик может это делать, главное чтобы изменения были логичными и обоснованны в описаниях к коммиту.

Начинаем кодить шаги. Думаю вы заметили что в обработке проверки поведения процедур проверки шагов меньше чем шагов в фиче. Этот функционал называется "повторное использование шагов", т.е. программируя шаг, нужно иметь в виду то что он может быть использован ещё много раз но уже с другими параметрами. Сначала позаботимся о контексте. Сейчас состояние шага "не реализован" и чтобы сделать шаг зеленым, т.е. пройденным нужно чтобы процедура соответствующая шагу отработала без исключительных ситуаций. Процедуру проверки шага нужно стараться использовать как точку входа в другие процедуры и функции в которых выполнять сравнение и вызывать исключения. Для такие процедур и функций есть специальное название "Ассерты", в

переводе asserts означает - Утверждения. Кроме того эти ассерты нужно стараться делать универсальными. Так же ассерты бывают простые (клиентские) и прикладные (выполняющиеся на стороне сервера). От слов к делу, а точнее к нашей процедуре проверки шага

```
&НаКлиенте
//Допустим я подготавливаю группу с наименованием "Материалы" справочника "Номенклатура"
//@ЯПодготавливаюГруппуСНаименованиемСправочника(Парам01,Парам02)
Процедура ЯПодготавливаюГруппуСНаименованиемСправочника(НаименованиеГруппы,
ПредставлениеСправочника) Экспорт
    Группа = утвПолучитьЭлементГруппуСправочника(ПредставлениеСправочника,, НаименованиеГруппы,
Истина);

    УдалитьЭлементыСправочникаВГруппе(ПредставлениеСправочника, Группа);
КонецПроцедуры
```

для начала, переименуем параметры, они были созданы автоматически, давайте дадим им более понятные имена: ПредставлениеСправочника, НаименованиеГруппы. Теперь займемся ассертом или утверждением, кому как удобно. По возможности шаги то же нужно стремиться делать независимыми, т.к. это не нагрузочное тестирование а разработка через поведение, мы можем позволить себе в каждом шаге выполнять поиск нужного элемента, документа и т.д. Для этого создадим некую универсальную функцию-утверждения

```
&НаСервере
Функция утвПолучитьЭлементГруппуСправочника(ИмяСправочника, Код = "", Наименование = "", ИскатьГруппу =
Ложь)
    ПредставлениеПоиска = "";

    Если Истина
        И ПустаяСтрока(Код)
        И ПустаяСтрока(Наименование)
        Тогда

            ВызватьИсключение "Не заполнено ни одно свойство поиска";

    ИначеЕсли Истина
        И ЗначениеЗаполнено(Код)
        И ЗначениеЗаполнено(Наименование)
        Тогда

            ПредставлениеПоиска = "коду " + Код + " и наименованию " + Наименование + ";

    ИначеЕсли ЗначениеЗаполнено(Код) Тогда
        ПредставлениеПоиска = "коду " + Код + ";

    ИначеЕсли ЗначениеЗаполнено(Наименование) Тогда
        ПредставлениеПоиска = "наименованию " + Наименование + ";

    КонецЕсли;

    ТекстИсключения = "Не нашли #ГруппуИлиЭлемент справочника #ИмяСправочника по
#ПредставлениеПоиска";

    ТекстИсключения = СтрЗаменить(ТекстИсключения, "#ГруппуИлиЭлемент", ?(ИскатьГруппу, "группу",
"элемент"));
    ТекстИсключения = СтрЗаменить(ТекстИсключения, "#ИмяСправочника", ИмяСправочника);
    ТекстИсключения = СтрЗаменить(ТекстИсключения, "#ПредставлениеПоиска", ПредставлениеПоиска);
```

```

Запрос = Новый Запрос;
Запрос.Текст =
"ВЫБРАТЬ
|      Справочник.Ссылка
|ИЗ
|      Справочник." + ИмяСправочника + " КАК Справочник
|ГДЕ
|      Справочник.ЭтоГруппа = &ИскатьГруппу
|      " + ?(ПустаяСтрока(Код), "", "И Справочник.Код = &Код") + "
|      " + ?(ПустаяСтрока(Наименование), "", "И Справочник.Наименование = &Наименование") + "
|";

Запрос.УстановитьПараметр("ИскатьГруппу", ИскатьГруппу);
Запрос.УстановитьПараметр("Код", Код);
Запрос.УстановитьПараметр("Наименование", Наименование);

РезультатЗапроса = Запрос.Выполнить();
Если РезультатЗапроса.Пустой() Тогда
    ВызватьИсключение ТекстИсключения;

КонецЕсли;

ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();
ВыборкаДетальныеЗаписи.Следующий();

Возврат ВыборкаДетальныеЗаписи.Ссылка;
КонецФункции

```

когда нужная группа номенклатуры найдена, передадим ее в процедуру которая удалит все элементы в этой группе

```

&НаСервере
Процедура УдалитьЭлементыСправочникаВГруппе(ИмяСправочника, Группа)
    Запрос = Новый Запрос;
    Запрос.Текст =
"ВЫБРАТЬ
|      Справочник.Ссылка
|ИЗ
|      Справочник." + ИмяСправочника + " КАК Справочник
|ГДЕ
|      Справочник.Родитель = &Группа";

    Запрос.УстановитьПараметр("Группа", Группа);

    РезультатЗапроса = Запрос.Выполнить();
    Если РезультатЗапроса.Пустой() Тогда
        Возврат;
    КонецЕсли;

    ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();
    Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
        ЭлементОбъект = ВыборкаДетальныеЗаписи.Ссылка.ПолучитьОбъект();

        ЭлементОбъект.Удалить();
    КонецЦикла;
КонецПроцедуры

```

Сохраним обработку, перезагрузим и выполним текущий сценарий чтобы выполнялся контекст, напомним контекст выполняется перед каждым сценарием. Шаг подготовки справочника отработал и причем без ошибок что с одной стороны хорошо, но с другой

лишает меня возможности познакомить вас с тонкостями отладки шагов. Но я все же расскажу вам о них и начнем с простого:

- Для отладки шагов можно разместить на форме команду и с ее помощью вызывать процедуру проверки шага. Т.к. на вход, т.е. через параметры в процедуру проверки шага мы передаем простые типы (строка, дата, число) то сложностей здесь нет

```
&НаКлиенте
Процедура Команда1(Команда)
    ИмяПроцедурыПроверкиШага(<ПараметрТипаСтрока>, <ПараметрТипаЧисло>, <ПараметрТипаДата>);
КонецПроцедуры
```

- Далее немного по сложнее. Через переменную “Ванесса” мы можем вызывать процедуры и функции из обработки vanessa-behavior. В создаваемые шаги этот вызов проставляется автоматически но выноситься в комментарий. Чтобы использовать это метод нужно раскомментировать метод и передать в него тот параметр который мы хотим проверить (посмотреть). Кстати, давайте посмотрим в каком виде behavior передает таблицу. Вынесем в комментарий код из всех шагов кроме последнего а так же нужно чтобы был открыт behavior и оставалось только нажать кнопку “Выполнить сценарий”. В такой, исходной ситуации, устанавливаем флаг “Остановка по ошибке” и запускаем выполнение сценария. Готово! Вот мы уже и в модуле формы behavior’a и можем посмотреть значение. Только прежде чем прекратить отладку не забывайте снимать флаг “Останавливаться по ошибке” и и раскомментировать шаги).
- И самый сложный вариант - это точка останова. Дело в том что если мы в шаге проверки поведения поставим точку останова - останова не произойдет. Но если мы пере откроем 1С предприятие и сначала откроем обработку проверки поведения, а потом vanessa-behavior, то точка останова сработает и можно будет пройти отладчиком по всем шагам.

Итак, с тонкостями отладки мы разобрались - возвращаемся к программированию шагов проверки, а точнее к шагу

```
Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" нет элементов
```

проверим этот шаг так

```
&НаКлиенте
//Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" нет элементов
//@ВСправочникеВГруппеСНаименованиемНетЭлементов(Парам01,Парам02)
Процедура ВСправочникеВГруппеСНаименованиемНетЭлементов(ПредставлениеСправочника,
НаименованиеГруппы) Экспорт
    Группа = утвПолучитьЭлементГруппуСправочника(ПредставлениеСправочника,, НаименованиеГруппы,
Истина);

    утвВГруппеСправочникаНетЭлементов(ПредставлениеСправочника, Группа, "В группе уже есть
материалы");
КонецПроцедуры
```

Найдем группу номенклатуры и передадим ее в прикладной ассерт проверяющий наличие элементов в группе

```
&НаСервере
Процедура утвВГруппеСправочникаНетЭлементов(ИмяСправочника, Группа, ТекстИсключения = "")
    Запрос = Новый Запрос;
    Запрос.Текст =
        "ВЫБРАТЬ
        |         Справочник.Ссылка КАК Ссылка
        |ИЗ
        |         Справочник." + ИмяСправочника + " КАК Справочник
        |ГДЕ
        |         Справочник.Родитель = &Группа";

    Запрос.УстановитьПараметр("Группа", Группа);

    РезультатЗапроса = Запрос.Выполнить();
    Если Не РезультатЗапроса.Пустой() Тогда
        ВызватьИсключение ТекстИсключения;
    КонецЕсли;
КонецПроцедуры
```

Сохраняем, проверяем. Чтобы реализовать следующие шаги воспользуемся функционалом “фичи из воздуха” а так же новым для нас функционалом “детализация шагов”. Начнем запись действий пользователя. Стоп, прекратим запись, т.е. у нас же ещё нет обработки для загрузки материалов из файла. Давай те ее создадим, хотя бы форму и основные элементы управления. Перейдем в конфигуратор и создадим обработку. Дадим имя обработке и сохраним ее. Т.к. мы начали разработку уже самого функционала то нужно сделать коммит с описанием “начал разработку” и обязательно указать номер issue. Создадим форму и разместим на ней команду и два поля “ПутьКФайлу” и “ВыбраннаяГруппа”, для поля “ПутьКФАйлу” ещё добавим кнопки выбора и очистки. Напишем код выбора файла

```
&НаКлиенте
Процедура ПутьКФайлуНачалоВыбора(Элемент, ДанныеВыбора, СтандартнаяОбработка)
    СтандартнаяОбработка = Ложь;

    ВыборФайла = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.Открытие);

    ВыборФайла.Показать(Новый ОписаниеОповещения("ПутьКФайлуНачалоВыбораЗавершение",
        ЭтотОбъект));
КонецПроцедуры

&НаКлиенте
Процедура ПутьКФайлуНачалоВыбораЗавершение(ВыбранныеФайлы, ДополнительныеПараметры) Экспорт
    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    ПутьКФайлу = ВыбранныеФайлы[0];
КонецПроцедуры
```

Готово, проверим что файл выбирается. Кстати, файл json положим в каталог examples, что в переводе означает “примеры”. Откроем обработку и выберем файл, все ок.

А вот теперь давайте подумаем как реализовать шаг “я открываю обработку”. Ведь в автоматизированном тестировании ещё нет возможность работать с системным диалогом. Да, мы можем прикрепить его дополнительным отчетам и обработкам, но на время отладки хотелось бы не делать лишних действий. Воспользуемся обработкой из продукта [vanessa-stack-commons](#). Выкачаем репозиторий продукта и откроем в клиенте тестирования обработку hotclick, как вы знаете есть горячие, т.е. быстро нажимаемые клавиши, а это быстро открываемые обработки. Используя эту обработку нужно учитывать что закрыть ее можно только по соответствующей кнопке, позже я поясню для чего это сделано. Выберем нашу обработку для загрузки файла а так же выберем ту форму обработки которую нужно открывать и вот теперь начнем запись действий пользователя.

Выберем нашу обработку и завершим запись действий пользователя. Для начала скопируем контекст и вставим его в нашу фичу. А затем скопируем шаги сценария и перенесем в нашу фичу. С помощью клавиши Tab сдвинем скопированные шаги и добавим тег @tree над ключевым словом “Функционал”. Сохраним фичу и перезагрузим ее в behavior. Видим что написанный нами шаг теперь выведен курсивом а шаги которые мы скопированы находятся в нашем шаге. Именно так работает функционал детализации шагов. Поясню, чтобы он работал нужен тег @tree и Tab. Чтобы видеть ход выполнения шагов и воспроизведение действий пользователя - растащим мастера и клиента тестирования по разным сторонам экрана и запустим выполнения сценария. Готово - обработка открылась, а мы и строчки кода не написали. Так же поступим и со следующим шагом. Начнем запись, выберем файл и закончим запись. Полученный шаг скопируем в нашу фичу и не забудем про Tab и сохраним фичу. Запустим и увидим что поле пустое. Дело в том что как я и говорил автоматизированное тестирование ещё не умеет работать с системными окнами, но в 1С смогли решить данную задачу другим способом, а именно с помощью метода “УстановитьРезультатДиалогаВыбораФайла”. Этот метод нужно вызвать до выполнения команды в результате которой появиться системный диалог и параметром этого метода нужно передать путь к файлу который мы бы выбрали. Когда сработает команда показа диалога то процедуру указанную в обработке оповещения будет передан путь к файлу, который мы указали в методе “УстановитьРезультатДиалогаВыбораФайла”. Выглядит это вот так:

```
&НаКлиенте
//И я указал путь к файлу
//@ЯУказалПутьКФайлу()
Процедура ЯУказалПутьКФайлу() Экспорт
    КонтекстСохраняемый.ТестовоеПриложение.УстановитьРезультатДиалогаВыбораФайла(Истина,
        "C:\repo\work\examples\materials.json");
КонецПроцедуры
```

но “жестко” указать путь мы не можем, т.к. на разных ПК будет запускаться фича, поэтому воспользуемся следующими процедурами

```
&НаКлиенте
Функция ПолучитьПутьКФайлуОтносительноКаталогаFeatures(ИмяФайлаСРасширением)
    ПутьКФайлу = "";
```

```

        СостояниеVanessaBehavior = Ванесса.ПолучитьСостояниеVanessaBehavior();

        ПутьКТекущемуFeatureФайлу = СостояниеVanessaBehavior.ТекущаяФича.ПолныйПуть;

        ПутьКФайлу = Лев(ПутьКТекущемуFeatureФайлу, СтрНайти(ПутьКТекущемуFeatureФайлу, "features") - 1) +
        ИмяФайлаСРасширением;

        Возврат ПутьКФайлу;
    КонецФункции

&НаКлиенте
//И я указал путь к фалу
//@ЯУказалПутьКФалу()
Процедура ЯУказалПутьКФалу() Экспорт
    КонтекстСохраняемый.ТестовоеПриложение.УстановитьРезультатДиалогаВыбораФайла(Истина,
        ПолучитьПутьКФайлОтносительноКаталогаFeatures("examples\materials.json"));
КонецПроцедуры

```

с шагами выбора группы номенклатуры и нажатия кнопки загрузить сложностей нет. Не смотря на то что следующий шаг упал давайте закоммитим промежуточный результат с описанием “реализовал секции: исходная и действие”. Теперь нам собственно нужен функционал который загрузит материалы

```

&НаКлиенте
Процедура Загрузить(Команда)
    АдресФайлаВоВременномХранилище = "";

    ДополнительныеПараметры = Новый Структура;

    ДополнительныеПараметры.Вставить("ВыбраннаяГруппа", ВыбраннаяГруппа);

    НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьДанныеЗавершение", ЭтотОбъект,
        ДополнительныеПараметры), АдресФайлаВоВременномХранилище, ПутьКФайлу, Ложь);
КонецПроцедуры

&НаКлиенте
Процедура ЗагрузитьДанныеЗавершение(Результат, Адрес, ВыбранноеИмяФайла, ДополнительныеПараметры)
    Экспорт
        Если Результат Тогда
            ЗагрузитьДанныеНаСервере(Адрес, ДополнительныеПараметры);
        КонецЕсли;
КонецПроцедуры

&НаСервере
Процедура ЗагрузитьДанныеНаСервере(Адрес, ДополнительныеПараметры)
    ДвоичныеДанные = ПолучитьИзВременногоХранилища(Адрес);
    ИмяВременногоФайла = ПолучитьИмяВременногоФайла();
    ДвоичныеДанные.Записать(ИмяВременногоФайла);

    ПрочитатьИЗагрузить(ИмяВременногоФайла, ДополнительныеПараметры);

    Файл = Новый Файл(ИмяВременногоФайла);

    Если Файл.Существует() Тогда
        УдалитьФайлы(ИмяВременногоФайла);
    КонецЕсли;
КонецПроцедуры

&НаСервере
Процедура ПрочитатьИЗагрузить(ИмяВременногоФайла, ДополнительныеПараметры)
    Читатель = Новый ЧтениеJSON;

```

```
Читатель.ОткрытьФайл(ИмяВременногоФайла);

НовыйМатериал = Неопределено;
НужноЗаписатьНовыйМатериал = Ложь;

Пока Читатель.Прочитать() Цикл
    Если Читатель.ТипТекущегоЗначения = ТипЗначенияJSON.ИмяСвойства Тогда
        Если Читатель.ТекущееЗначение = "material" Тогда
            НужноЗаписатьНовыйМатериал = Истина;

            НовыйМатериал = Справочники.Номенклатура.СоздатьЭлемент();

            НовыйМатериал.Родитель = ДополнительныеПараметры.ВыбраннаяГруппа;

        ИначеЕсли Читатель.ТекущееЗначение = "key" Тогда
            Читатель.Прочитать();

            НовыйМатериал.Код = Читатель.ТекущееЗначение;

        ИначеЕсли Читатель.ТекущееЗначение = "name" Тогда
            Читатель.Прочитать();

            НовыйМатериал.Наименование = Читатель.ТекущееЗначение;

        КонецЕсли;

    ИначеЕсли Истина
        И Читатель.ТипТекущегоЗначения = ТипЗначенияJSON.КонецОбъекта
        И НужноЗаписатьНовыйМатериал
        Тогда

            НужноЗаписатьНовыйМатериал = Ложь;

            ИскомыйМатериал = Справочники.Номенклатура.НайтиПоКоду(НовыйМатериал.Код);
            Если Ложь
                Или ИскомыйМатериал = Неопределено
                Или ИскомыйМатериал = Справочники.Номенклатура.ПустаяСсылка()
                Тогда

                    НовыйМатериал.Записать();

                КонецЕсли;
            КонецЕсли;
        КонецЦикла;

    Читатель.Закрыть();
КонецПроцедуры
```

Проверять успешность загрузки будет опять же с помощью функционала “фичи из “воздуха” и в итоге получим такой текст:

```
# encoding: utf-8
# language: ru
```

```
@tree
```

Функционал: Загрузка материалов

Как бухгалтер

Хочу внешнюю обработку для загрузки материалов из файла

Чтобы экономить время и исключить ошибки заполнения

Контекст

Допустим я подготавливаю группу с наименованием "Материалы" справочника "Номенклатура"

И Я запускаю сценарий открытия **TestClient** или подключаю уже существующий

И Пауза 3

Сценарий: Загрузка материалов из файла materials.json

Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" нет элементов
Когда я открываю обработку загрузки материалов

Когда открылось окно "hot click"

И В форме "hot click" в таблице "Состав" я перехожу к строке:

| 'Имя' |

| 'ЗагрузкаМатериаловИзФайла' |

И В открытой форме я нажимаю на кнопку "Открыть обработку"

И я выбираю файл

Когда я указываю путь к файлу

И открылось окно "Загрузка материалов из файла"

И В открытой форме я нажимаю кнопку выбора у поля "Путь к файлу"

И я выбираю группу с наименованием "Материалы"

Когда открылось окно "Загрузка материалов из файла"

И В открытой форме я открываю выпадающий список "Группа загрузки"

И В открытой форме я выбираю значение реквизита "Группа загрузки" из формы списка

Тогда открылось окно "Номенклатура"

И В форме "Номенклатура" в таблице "Список" я перехожу к строке:

| 'Наименование' |

| 'Материалы' |

И В открытой форме я нажимаю на кнопку "Выбрать"

И я нажимаю кнопку загрузить

Когда открылось окно "Загрузка материалов из файла"

И В открытой форме я нажимаю на кнопку "Загрузить"

Тогда материалы загрузились корректно

Когда В панели разделов я выбираю "Справочники"

И В панели функций я выбираю "Номенклатура"

И открылось окно "Номенклатура"

И В форме "Номенклатура" в таблице "Список" я перехожу к строке:

| 'Наименование' |

| 'Материалы' |

И В форме "Номенклатура" в ТЧ "Список" я выбираю текущую строку

И таблица формы с именем "Список" стала равной:

| 'Код' | 'Наименование' |

| '00-00000007' | 'Материалы' |

| '00002' | 'Складывающиеся лопасти воздушного винта CAMcarbon 9,5 x 5' |

| '00003' | 'Ступица воздушного винта 32/3/0' |

| '00001' | 'Электродвигатель Hacker A30-14M V2 6-Pole 6.7:1 (brushless motor)' |

Сценарий: Загрузка материалов из файла materials.json с проверкой синхронизации

Допустим в справочнике "Номенклатура" в группе с наименованием "Материалы" есть элементы

Когда В панели разделов я выбираю "Справочники"

И В панели функций я выбираю "Номенклатура"

И открылось окно "Номенклатура"

И В форме "Номенклатура" в таблице "Список" я перехожу к строке:

| 'Наименование' |

| 'Материалы' |

И В форме "Номенклатура" в ТЧ "Список" я выбираю текущую строку

И таблица формы с именем "Список" стала равной:

| 'Код' | 'Наименование' |

| '00-00000007' | 'Материалы' |

| '00002' | 'Складывающиеся лопасти воздушного винта CAMcarbon 9,5 x 5' |

| '00001' | 'Электродвигатель Hacker A30-14M V2 6-Pole 6.7:1 (brushless motor)' |

И Я закрываю окно "Номенклатура"

И Пауза 5

Когда я открываю обработку загрузки материалов

Когда открылось окно "hot click"

И В форме "hot click" в таблице "Состав" я перехожу к строке:

| 'Имя' |

| 'ЗагрузкаМатериаловИзФайла' |

И В открытой форме я нажимаю на кнопку "Открыть обработку"

```

И я выбираю файл
  Когда я указываю путь к файлу
  И открылось окно "Загрузка материалов из файла"
  И В открытой форме я нажимаю кнопку выбора у поля "Путь к файлу"

И я выбираю группу с наименованием "Материалы"
  Когда открылось окно "Загрузка материалов из файла"
  И В открытой форме я открываю выпадающий список "Группа загрузки"
  И В открытой форме я выбираю значение реквизита "Группа загрузки" из формы списка
  Тогда открылось окно "Номенклатура"
  И В форме "Номенклатура" в таблице "Список" я перехожу к строке:
  | 'Наименование' |
  | 'Материалы' |
  И В открытой форме я нажимаю на кнопку "Выбрать"

И я нажимаю кнопку загрузить
  Когда открылось окно "Загрузка материалов из файла"
  И В открытой форме я нажимаю на кнопку "Загрузить"

Тогда материалы загрузились корректно
  Когда В панели разделов я выбираю "Справочники"
  И В панели функций я выбираю "Номенклатура"
  И открылось окно "Номенклатура"
  И В форме "Номенклатура" в таблице "Список" я перехожу к строке:
  | 'Наименование' |
  | 'Материалы' |
  И В форме "Номенклатура" в ТЧ "Список" я выбираю текущую строку
  И таблица формы с именем "Список" стала равной:
      | 'Код' | | 'Наименование' | |
      | '00-00000007' | | 'Материалы' | |
      | '00002 ' | | 'Складывающиеся лопасти воздушного винта CAMcarbon 9,5 x 5' | |
      | '00003 ' | | 'Ступица воздушного винта 32/3/0' | |
      | '00001 ' | | 'Электродвигатель Hacker A30-14M V2 6-Pole 6.7:1 (brushless motor)' |

```

Нам осталось научиться подготавливать инфобазу, а точнее справочник “Номенклатура” для прохождения второго сценария. Напомню на момент выполнения первого шага сценария в группе справочника должны быть материалы, чтобы мы могли проверить корректность загрузки. В этом нам поможет такое понятие как “fixtures”, т.е. в противоположность “feature” (будущие данные, функционал) уже существующие данные.

Для работы с fixtures нам потребуется [обработка](http://its.1c.ru/) с сайта <http://its.1c.ru/>. Платная подписка не потребуется, но т.к. 1С является исключительным правообладателем этой обработки, создадим в корне нашего репозитория файл с именем NOTICE и расширением md. И в этом файле напишем

```
Обработка портирована с сайта ИТС (http://its.1c.ru/db/metod8dev#content:4126:hdoc)
```

Удалим один из элементов в группе “Материалы” и с помощью обработки “ВыгрузкаЗагрузкаДанныхXML” получим текст файла выгрузки. В обработке проверки поведения создадим текстовый документ и скопируем в него текст файла выгрузки. Сохраним изменения и сделаем коммит с описанием “Создал макет fixtures”. Теперь доработаем шаг контекста в котором мы подготавливали справочник

```

&НаКлиенте
//Допустим я подготавливаю группу с наименованием "Материалы" справочника "Номенклатура"
//@ЯПодготавливаюГруппуСНаименованиемСправочника(Парам01,Парам02)

```

```

Процедура ЯПодготавливаюГруппуСНаименованиемСправочника(НаименованиеГруппы,
ПредставлениеСправочника) Экспорт
    Группа = утвПолучитьЭлементГруппыСправочника(ПредставлениеСправочника,, НаименованиеГруппы,
Истина);

    УдалитьЭлементыСправочникаВГруппе(ПредставлениеСправочника, Группа);

    СостояниеVanessaBehavior = Ванесса.ПолучитьСостояниеVanessaBehavior();
    Если СостояниеVanessaBehavior.ТекущийСценарий.Имя = "Загрузка материалов из файла materials.json с
проверкой синхронизации" Тогда
        ЗагрузитьFixtureИзМакета("МатериалыИзФайла");
    КонецЕсли;
КонецПроцедуры

```

Если шаги контекста вызваны вторым сценарием, мы вызываем метод загрузки фикстур и передаем в него имя того макета который хотим загрузить

```

&НаКлиенте
Процедура ЗагрузитьFixtureИзМакета(ИмяМакета)
    Ванесса.ЗапретитьВыполнениеШагов();

    НачальноеИмяФайла = ПолучитьПутьКФайлОтносительноКаталогаFeatures("tools\Выгрузка и загрузка
данных XML.erf");

    Адрес = "";

    НачатьПомещениеФайла(Новый ОписаниеОповещения("ЗагрузитьFixtureИзМакетаЗавершение",
ЭтотОбъект, ИмяМакета), Адрес, НачальноеИмяФайла, Ложь);
КонецПроцедуры

&НаКлиенте
Процедура ЗагрузитьFixtureИзМакетаЗавершение(УдалосьПоместитьФайл, Адрес, ВыбранноеИмяФайла,
ИмяМакета) Экспорт
    ЗагрузитьFixtureИзМакетаЗавершениеНаСервере(Адрес, ИмяМакета);

    Ванесса.ПродолжитьВыполнениеШагов();
КонецПроцедуры

&НаСервере
Процедура ЗагрузитьFixtureИзМакетаЗавершениеНаСервере(Адрес, ИмяМакета)
    ИмяВременногоФайла = ПолучитьИмяВременногоФайла();

    ДвоичныеДанные = ПолучитьИзВременногоХранилища(Адрес);
    ДвоичныеДанные.Записать(ИмяВременногоФайла);

    ВнешняяОбработка = ВнешниеОбработки.Создать(ИмяВременногоФайла, Ложь);

    ИмяВременногоФайла = ПолучитьИмяВременногоФайла();

    Текст = РеквизитФормыВЗначение("Объект").ПолучитьМакет(ИмяМакета).ПолучитьТекст();

    ВременныйДокумент = Новый ТекстовыйДокумент;
    ВременныйДокумент.УстановитьТекст(Текст);
    ВременныйДокумент.Записать(ИмяВременногоФайла, КодировкаТекста.UTF8);

    ВнешняяОбработка.ВыполнитьЗагрузку(ИмяВременногоФайла);
КонецПроцедуры

```

Т.к. мы используем асинхронный метод, то возникает необходимость приостановить выполнения шагов и когда асинхронный метод отработает - снова запустить выполнение шагов. Для этого есть два соответствующих метода

```
Ванесса.ЗапретитьВыполнениеШагов();
```

```
Ванесса.ПродолжитьВыполнениеШагов();
```

Перезагрузим фичу и выполним сценарий. Готово!

Конфигурация с возможностью изменения

Представим что мы получили от заказчика письмо в котором он просит оптимизировать его работу с некоторыми материалами.

Первое что мы сделаем - это создадим задачу в которой зафиксируем исходное требование и к которой будем привязывать все наши коммиты.

Далее, “идем” общаться с заказчиком. Выясняем что нужен функционал, который в момент поступления товара, а точнее материала на субсчета 10-го счета, спишет тот поступивший материалы на затраты, который выбран в специальном регистре сведений. Давайте опишем этот функционал в фиче. Откроем bddeditor и создадим новый feature файл. Функционал назовем “Автоматическое списание канцелярии”. Пользовательская история будет такая:

```
Как бухгалтер
Хочу автоматического списания канцелярии в момент оприходования
Чтобы не заниматься рутиной
```

На первый взгляд нужно проверить что материал которого нет в “специальном” регистре - проводится корректно и не списывается на счет затрат, а то материал который выбран - проводится корректно и списывается на счет затрат, разумеется корректно. Несколько сценариев создавать не потребуется, мы в табличной части выберем материал который есть и которого нет в “специальном” регистре сведений. Итак, итоговая фича будет выглядеть так:

```
# encoding: utf-8
# language: ru
```

```
@tree
```

```
Функционал: Автоматическое списание канцелярии
  Как бухгалтер
  Хочу автоматического списания канцелярии в момент оприходования
  Чтобы не заниматься рутиной
```

```
Контекст:
  Дано Я запускаю сценарий открытия TestClient или подключаю уже существующий
```

```
Сценарий: Поступление материалов
  Допустим я подготавливаю вспомогательные данные
  Когда я создаю поступление товаров и услуг
  И я заполняю шапку документа
  И я выбираю материалы
  И я провожу документ
  Тогда результат проведения корректен
```

Закоммитим с описанием “#2 составили фичу”. Без доработки конфигурации такой функционал не реализовать, по этому включим возможность изменения конфигурации.

Создадим подписку на событие. Назовем ее “sb_ПроведениеДокументов” в качестве источника выберем “ДокументОбъект.ПоступлениеТоваровУслуг”, событие “ОбработкаПроведения”. Для обработчика создадим общий модуль “sb_ДополнениеПроведения” и создадим в нем процедуру обработчик.

```
Процедура sb_ПроведениеДокументовОбработкаПроведения(Источник, Отказ, РежимПроведения) Экспорт
КонецПроцедуры
```

Ещё нам потребуется регистр сведений, чтобы пользователь мог выбрать те материалы, которые должны в момент поступления автоматически списываться на счет затрат. Итак, создадим регистр сведений с именем “sb_МатериалыИСчетаЗатрат”. Регистр будет независимым и непериодическим, а его данные мы создадим таким образом, чтобы для одного материала можно было указать только одно соответствие счета затрат и субконто затрат. Значит измерения

- Материал, тип справочник ссылка Номенклатура
- и два ресурса
 - СчетЗатрат, тип план счетов ссылка Хозрасчетный
 - СубконтоЗатрат1, тип справочник ссылка СтатьиЗатрат

Сохраним конфигурацию и займемся дополнением обработки проведения. У меня получился следующий алгоритм автоматического списания канцелярии:

```
Процедура sb_ПроведениеДокументовОбработкаПроведения(Источник, Отказ, РежимПроведения) Экспорт
    Если Отказ Тогда
        Возврат;
    КонецЕсли;

    Если ТипЗнч(Источник) = Тип("ДокументОбъект.ПоступлениеТоваровУслуг") Тогда
        СписатьМатериалы(Источник);
    КонецЕсли;
КонецПроцедуры

Процедура СписатьМатериалы(Источник)
    ТаблицаДанных = ПолучитьПустуюТаблицуДанных();

    Счет_10 = ПланыСчетов.Хозрасчетный.Материалы;

    ДвиженияБухучет = ПолучитьДвиженияБухучет(Источник);
    Для Каждого Движение Из ДвиженияБухучет Цикл
        Если Движение.СчетДт.ПринадлежитЭлементу(Счет_10) Тогда
            ДобавитьСтрокуДанныхМатериалов(ТаблицаДанных, Движение);
        КонецЕсли;
    КонецЦикла;

    Если ТаблицаДанных.Количество() = 0 Тогда
        Возврат;
    КонецЕсли;

    ОпределитьСчетаИАналитикуЗатрат(ТаблицаДанных);
    ВыполнитьСписаниеМатериалов(ДвиженияБухучет, ТаблицаДанных);
КонецПроцедуры
```

Функция ПолучитьПустуюТаблицуДанных()

ТаблицаДанных = Новый ТаблицаЗначений;

ТаблицаДанных.Колонки.Добавить("Период");
ТаблицаДанных.Колонки.Добавить("Организация");
ТаблицаДанных.Колонки.Добавить("СчетКт");
ТаблицаДанных.Колонки.Добавить("СубконтоКт");
ТаблицаДанных.Колонки.Добавить("СчетДт");
ТаблицаДанных.Колонки.Добавить("СубконтоДт");
ТаблицаДанных.Колонки.Добавить("Сумма");
ТаблицаДанных.Колонки.Добавить("КоличествоКт");

Возврат ТаблицаДанных;

КонецФункции

Функция ПолучитьДвиженияБухучет(Источник)

ДвиженияБухучет = Неопределено;

Для Каждого Движение Из Источник.Движения Цикл

Если ТипЗнч(Движение) = Тип("РегистрБухгалтерииНаборЗаписей.Хозрасчетный") Тогда
ДвиженияБухучет = Источник.Движения.Хозрасчетный;

Прервать;

КонецЕсли;

КонецЦикла;

Возврат ДвиженияБухучет;

КонецФункции

Процедура ДобавитьСтрокуДанныхМатериалов(ТаблицаДанных, Движение)

НоваяСтрока = ТаблицаДанных.Добавить();

НоваяСтрока.СубконтоКт = Движение.СубконтоДт.Номенклатура;
НоваяСтрока.Период = Движение.Период;
НоваяСтрока.Организация = Движение.Организация;
НоваяСтрока.СчетКт = Движение.СчетДт;
НоваяСтрока.Сумма = Движение.Сумма;
НоваяСтрока.КоличествоКт = Движение.КоличествоДт;

КонецПроцедуры

Процедура ОпределитьСчетаИАналитикуЗатрат(ТаблицаДанных)

Запрос = Новый Запрос;

Запрос.Текст =

"ВЫБРАТЬ

| sb_МатериалыИСчетаЗатрат.Материал,

| sb_МатериалыИСчетаЗатрат.СчетЗатрат,

| sb_МатериалыИСчетаЗатрат.СубконтоЗатрат1

| ИЗ

| РегистрСведений.sb_МатериалыИСчетаЗатрат КАК sb_МатериалыИСчетаЗатрат

| ГДЕ

| sb_МатериалыИСчетаЗатрат.Материал В(&Материал)";

Запрос.УстановитьПараметр("Материал", ТаблицаДанных.ВыгрузитьКолонку("СубконтоКт"));

РезультатЗапроса = Запрос.Выполнить();

Если РезультатЗапроса.Пустой() Тогда

Возврат;

КонецЕсли;

ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();

Пока ВыборкаДетальныеЗаписи.Следующий() Цикл

ИскомаяСтрока = ТаблицаДанных.Найти(ВыборкаДетальныеЗаписи.Материал, "СубконтоКт");

Если ИскомаяСтрока = Неопределено Тогда

Продолжить;

КонецЕсли;

ИскомаяСтрока.СчетДт = ВыборкаДетальныеЗаписи.СчетЗатрат;

```
        ИскомаяСтрока.СубконтоДт = ВыборкаДетальныеЗаписи.СубконтоЗатрат1;
    КонецЦикла;
КонецПроцедуры

Процедура ВыполнитьСписаниеМатериалов(ДвиженияБухучет, ТаблицаДанных)
    СписалиМатериалы = Ложь;

    Для Каждого СтрокаДанных Из ТаблицаДанных Цикл
        Если Истина
            И СтрокаДанных.СчетДт = Неопределено
            И СтрокаДанных.СубконтоДт = Неопределено
            Тогда

                Продолжить;
            КонецЕсли;

            Проводка = ДвиженияБухучет.Добавить();

            ЗаполнитьЗначенияСвойств(Проводка, СтрокаДанных);

            БухгалтерскийУчет.УстановитьСубконто(Проводка.СчетДт, Проводка.СубконтоДт, "СтатьиЗатрат",
            СтрокаДанных.СубконтоДт);

            БухгалтерскийУчет.УстановитьСубконто(Проводка.СчетКт, Проводка.СубконтоКт, "Номенклатура",
            СтрокаДанных.СубконтоКт);

            СписалиМатериалы = Истина;
        КонецЕсли;
    КонецЦикла;

    Если СписалиМатериалы Тогда
        ДвиженияБухучет.Записать();
    КонецЕсли;
КонецПроцедуры
```

Нам лишь остается с помощью функционалов “фичи из “воздуха” и “детализация шагов” доработать фичу. В итоге она будет выглядеть так:

```
# encoding: utf-8
# language: ru

@tree

Функционал: Автоматическое списание канцелярии
    Как бухгалтер
    Хочу автоматического списания канцелярии в момент оприходования
    Чтобы не заниматься рутинной

Контекст:
    Дано Я запускаю сценарий открытия TestClient или подключаю уже существующий

Сценарий: Поступление материалов
    Допустим я подготовливаю вспомогательные данные
    Когда я создаю поступление товаров и услуг
        Когда В панели разделов я выбираю "Покупки"
        И В панели функций я выбираю "Поступление (акты, накладные)"
        Тогда открылось окно "Поступление (акты, накладные)"
        И В открытой форме я нажимаю на кнопку "Товары (накладная)"
        Тогда открылось окно "Поступление товаров: Накладная (создание)"

        И я заполняю шапку документа
        И В открытой форме я открываю выпадающий список "Контрагент"
        И В открытой форме я выбираю значение реквизита "Контрагент" из формы списка
        #И В открытой форме я нажимаю кнопку выбора у поля "Контрагент"
        Тогда открылось окно "Контрагенты"
```


И В форме "Контрагенты" в таблице "Список" я перехожу к строке:
 | 'Наименование' | 'Полное наименование' |
 | 'Контрагент1' | 'Контрагент1' |
 И В форме "Контрагенты" в ТЧ "Список" я выбираю текущую строку

И я выбираю материалы
 Тогда открылось окно "Поступление товаров: Накладная (создание) **"
 И В открытой форме в ТЧ "Товары" я нажимаю на кнопку "Добавить"
 И В открытой форме в ТЧ "Товары" я выбираю значение реквизита "Номенклатура" из формы

списка

#И В открытой форме в ТЧ "Товары" я нажимаю кнопку выбора у реквизита "Номенклатура"

Тогда открылось окно "Номенклатура: Товары"
 И В форме "Номенклатура: Товары" в таблице "Список" я перехожу к строке:
 | '% НДС' | 'Единица' | 'Наименование' |
 | '18%' | 'шт' | 'Материал для автоматического списания' |
 И В форме "Номенклатура: Товары" в ТЧ "Список" я выбираю текущую строку
 Тогда открылось окно "Поступление товаров: Накладная (создание) **"
 И В открытой форме в ТЧ "Товары" я активизирую поле "Количество"
 И В открытой форме в ТЧ "Товары" в поле "Количество" я ввожу текст "10,000"
 И В открытой форме в ТЧ "Товары" я активизирую поле "Цена"
 И В открытой форме в ТЧ "Товары" в поле "Цена" я ввожу текст "100,00"
 И В форме "Поступление товаров: Накладная (создание) *" в ТЧ "Товары" я завершаю

редактирование строки

И В открытой форме в ТЧ "Товары" я нажимаю на кнопку "Добавить"
 И В открытой форме в ТЧ "Товары" я выбираю значение реквизита "Номенклатура" из формы

списка

#И В открытой форме в ТЧ "Товары" я нажимаю кнопку выбора у реквизита "Номенклатура"

Тогда открылось окно "Номенклатура: Товары"
 И В форме "Номенклатура: Товары" в таблице "Список" я перехожу к строке:
 | '% НДС' | 'Единица' | 'Наименование' |
 | '18%' | 'шт' | 'Материал НЕ для автоматического списания' |
 И В форме "Номенклатура: Товары" в ТЧ "Список" я выбираю текущую строку
 Тогда открылось окно "Поступление товаров: Накладная (создание) **"
 И В открытой форме в ТЧ "Товары" я активизирую поле "Количество"
 И В открытой форме в ТЧ "Товары" в поле "Количество" я ввожу текст "15,000"
 И В открытой форме в ТЧ "Товары" я активизирую поле "Цена"
 И В открытой форме в ТЧ "Товары" в поле "Цена" я ввожу текст "200,00"
 И В форме "Поступление товаров: Накладная (создание) *" в ТЧ "Товары" я завершаю

редактирование строки

И я провожу документ

Когда открылось окно "Поступление товаров: Накладная (создание) **"
 И В открытой форме я нажимаю на кнопку "Провести"
 И Пауза 10

Тогда результат проведения корректен

Когда открылось окно "Поступление товаров: Накладная **"
 И В открытой форме я нажимаю на кнопку "Движения документа"
 Тогда элемент формы с именем "РучнаяКорректировка" стал равен "Нет"
 И таблица формы с именем "Хозрасчетный" стала равной:

'Субконто2 Кт'	'Валюта Дт'	'Кол.'	'Субконто Кт'	'Сумма'	'Субконто3 Дт'	'Вал. сумма Дт'	'Содержание'
'Количество Дт'	'Подразделение Дт'	'Валюта Кт'	'Субконто3 Кт'				
'Количество Кт'	'Вал. сумма Кт'	'Подразделение Кт'					
'1'	'Материал для автоматического списания'			'10.01'	'27.04.2015'	'60.01'	
'Без договора'	"	'Контрагент1'	'1 000,00'	"	"	'10,000'	"
'*'	'Поступление материалов по вх.д. от'	"	"	"	"	"	"
'2'	'Материал НЕ для автоматического списания'	"	"	'10.01'	'27.04.2015'	'60.01'	
'Без договора'	"	'Контрагент1'	'3 000,00'	"	"	'15,000'	"
'*'	'Поступление материалов по вх.д. от'	"	"	"	"	"	"
'3'	'Прочие затраты'	"	"	'26'	'27.04.2015'	'10.01'	"
'Кол.'	'Материал для автоматического списания'	'1 000,00'	"	"	"	"	"
"	'10,000'	"	"	"	"	"	"

Закоммитим с описанием "#2 детализировал фичу".

Разумеется, нам потребуются фикстуры:

- Материалы
- Контрагент
- МатериалыИСчетаЗатрат

Для этого воспользуемся процедурами из прошлого кейса, а создавать макеты с текстом файла выгрузки мы умеем. Перезагрузим фичу и выполним. Готово!
Закоммитим с описанием “#2 реализовал и проверил функционал”.

Итоги

Итак, мы на практических примерах посмотрели как вести разработку по промышленным стандартам с использованием подхода к разработке через поведение.

Финальное задание

В рамках финального задания нужно написать скрипт, который нужно запускать из каталога который вы собираетесь сделать репозиторием. Т.е. в этом каталоге нужно:

- инициализировать git
- настроить пользователя и работу с кириллическими наименованиями файлов
- подключить precommit1C
- создать файлы и каталоги по bootstrap

Заключение

Про наработки и open source

В заключении обращаю ваше внимание на то что все продукты сообщества [silverbulleters](#) с открытым исходным кодом. Для закрепления материалов курса рекомендую изучить содержимое репозитория этих продуктов и сами продукты.