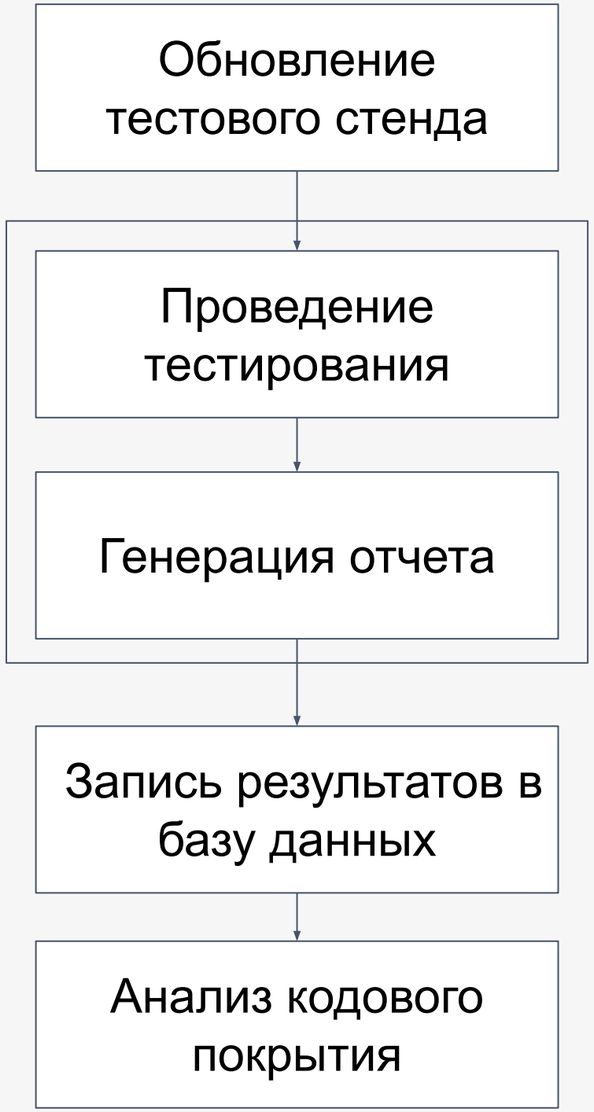
СВД
Встраиваемые
Системы

Методы автоматизации тестирования ПО

Анастасия Маштакова

Руководитель группы тестирования
Отдел автоматизации и верификации

Общий обзор: процессы автоматизации

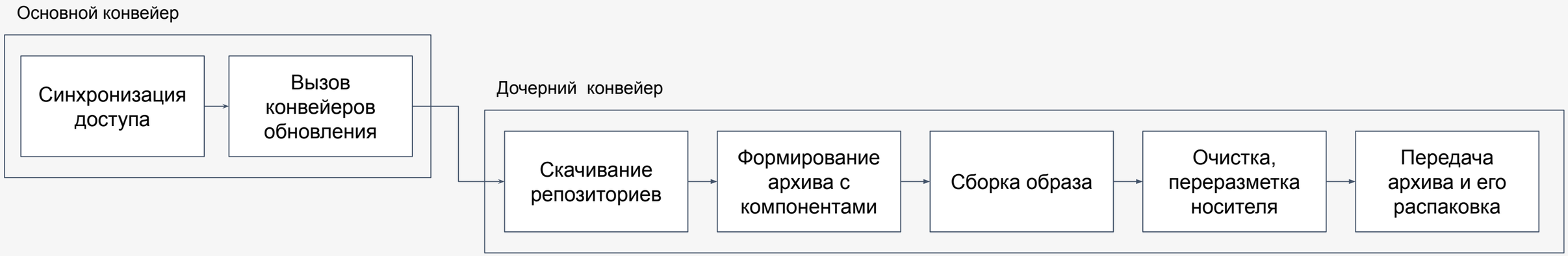


- Автоматизация включает в себя подготовку эталонного окружения, проведение тестирования, генерацию отчета, запись результатов и анализ кодового покрытия
- Эталонное окружение - обновленные до последней версии ОС устройства и docker-контейнеры
- В данном мастер-классе детально будет затронуто только проведение тестирования и формирование отчета

Общий обзор: описание инфраструктуры

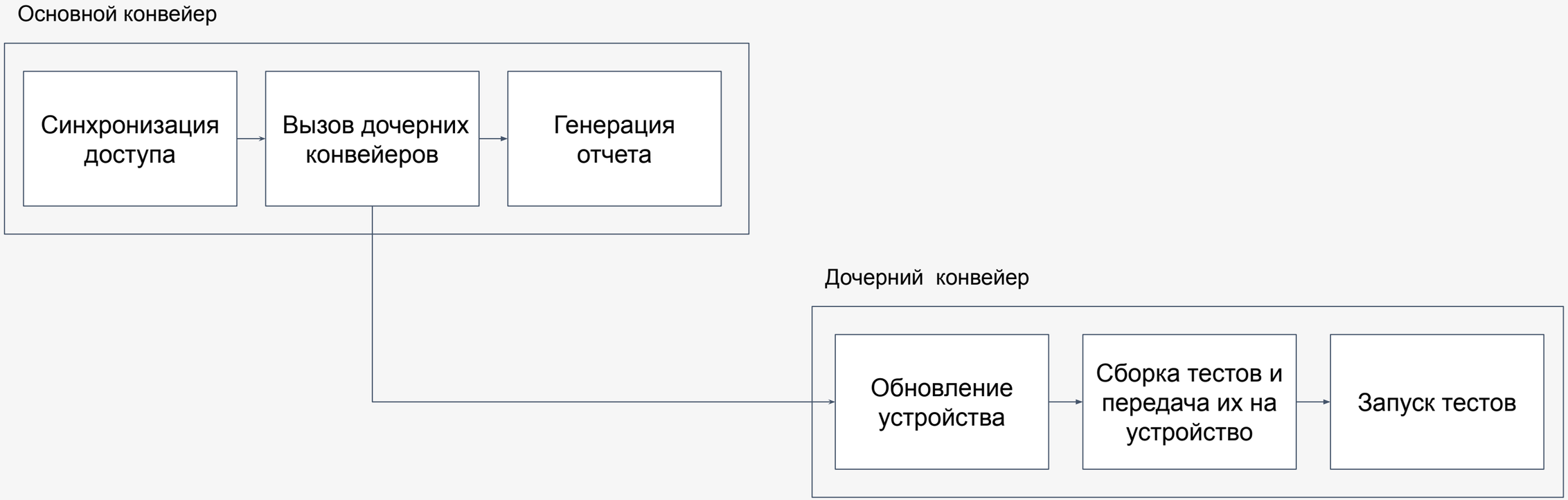
- Тестовый стенд - устройства на базе разных процессорных архитектур в одной подсети
- Загрузочный сервер - boot-сервер, хранящий загрузочные образы устройств тестового стенда
- Сборочный сервер - месторасположение инструмента CI/CD Jenkins, где также запускается соответствующий docker-образ

Общий обзор: принцип работы обновления



- Первый конвейер необходим для синхронизации доступа и более удобного запуска обновления нескольких устройств
- Второй производит непосредственно обновление устройства

Общий обзор: принцип работы тестирования



Общий обзор: общий вид конвейеров

Основной конвейер

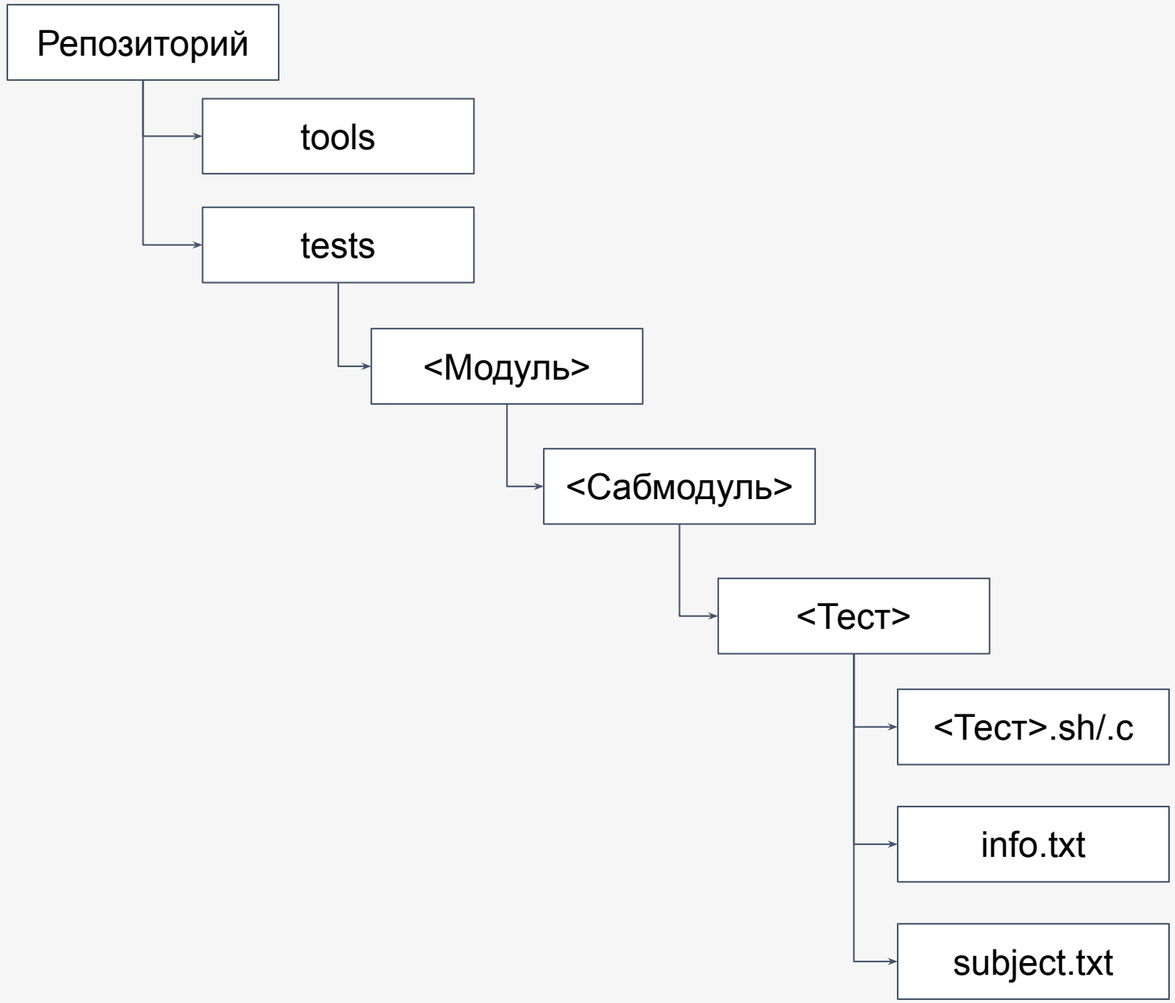
Stage View

	Declarative: Checkout SCM	Preparing	Testing	Analysis: docs & specs	PDF generating	Test: e2kle	Test:mipsbe	Test: armle-v7	Test: armle	Test: aarch64le-zynq	Test: ppcbe	Test: x86	Test: aarch64	Reporting	Declarative: Post Actions
Average stage times: (Average full run time: ~1h 44min)	3s	3min 15s	54ms	7min 58s	0ms	0ms	5h 19min	2min 18s	0ms	0ms	0ms	48min 48s	0ms	8min 48s	1s
#3114 Sep 21 14:10 No Changes	1s	7s	54ms	5s			5h 19min (paused for 1ms)							8min 48s	320ms
#3112 Sep 21 13:01 No Changes	6s	4min 59s	54ms	9s								48min 48s (paused for 2ms)		18min 15s	3s

Дочерний конвейер

	Declarative: Checkout SCM	Preparing	Upgrade	Checking	Building	Testing	Postprocessing	Declarative: Post Actions
Average stage times: (Average full run time: ~1h 4min)	6s	7s	9min 2s	15s	5s	5h 9min	15s	570ms
#3440 Sep 21 14:10 No Changes	1s	5s	9min 2s	2s	27s	5h 9min	15s	514ms
#3439 Sep 21 13:07 No Changes	11s	6s	4min 42s	5s	3s	43min 13s	15s	567ms

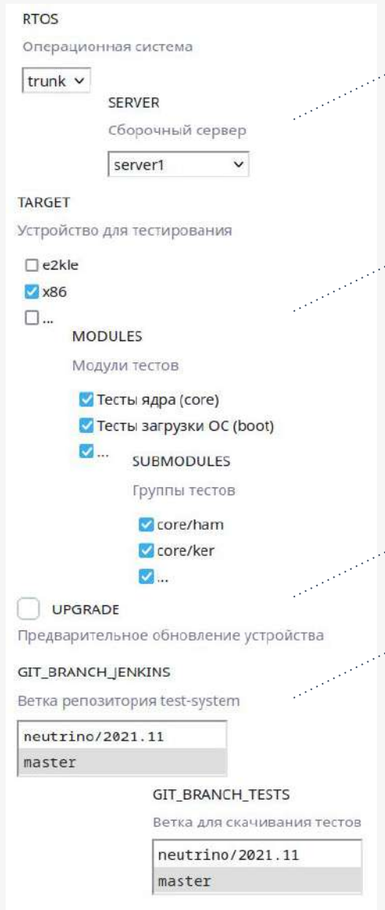
Общий обзор: иерархия тестов



- Тесты имеют иерархию <Модуль>/<Сабмодуль>/<Тест>
- Написаны на языках shell, Си
- В корне репозитория располагаются доп.файлы (скрипты, библиотеки)
- Рядом с тестом находится его краткое описание (info.txt) и перечень тестируемых компонентов (subject.txt)
- Для сборки используется рекурсивная сборочная система

Тестирование: параметры конвейера

- 4 вида параметров: extendedChoice (PT_SINGLE_SELECT), extendedChoice (PT_CHECKBOX), booleanParam, gitParameter




```
extendedChoice(  name: 'SERVER',
                 description: 'Сборочный сервер',
                 type: 'PT_SINGLE_SELECT',
                 multiSelectDelimiter: ',',
                 value: 'server1,...',
                 defaultValue: 'server1' )

extendedChoice(  name: 'MODULES',
                 description: 'Модули тестов',
                 type: 'PT_CHECKBOX',
                 visibleItemCount: TEST_MODULES_SZ,
                 multiSelectDelimiter: ',',
                 value: TEST_MODULES,
                 defaultValue: TEST_MODULES_DEFAULT,
                 descriptionPropertyValue: TEST_MODULES_DESCR )

booleanParam (   name: 'UPGRADE',
                 defaultValue: false,
                 description: 'Предварительное обновление устройства')

gitParameter( branch: '',
              branchFilter: 'origin/(.*)',
              defaultValue: 'master',
              description: 'Ветка репозитория test-system',
              name: 'GIT_BRANCH_JENKINS',
              quickFilterEnabled: false,
              selectedValue: 'DEFAULT',
              sortMode: 'NONE',
              tagFilter: '*',
              type: 'PT_BRANCH',
              useRepository: '.*tests/test-system.git' )
```

```
def TEST_MODULES          = "core," +
                            "boot," +
                            "..."

def TEST_MODULES_DESCR    = "Тесты ядра (core)," +
                            "Тесты загрузки ОС (boot)," +
                            "..."

def TEST_MODULES_DEFAULT  = TEST_MODULES
def TEST_MODULES_SZ       = TEST_MODULES.split(",").size()
```

Тестирование: основной конвейер

```
stage('Testing') {
    parallel {
        stage('Test: e2kle') {
            when { expression { return TARGET_LIST.contains("e2kle") } }
            steps {
                script {
                    run_testing( "e2kle", RUN_MODULES, RUN_SUBMODULES )
                }
            }
        }
        stage('Test: x86') {
            when { expression { return TARGET_LIST.contains("x86") } }
            steps {
                script {
                    run_testing( "x86", RUN_MODULES, RUN_SUBMODULES )
                }
            }
        }
...
```

- Тестирование разных устройств вызывается параллельно
- Монопольный доступ к устройству задается с помощью плагина Lockable Resources

```
def run_testing( String DEVICE, String RUN_MODULES, String RUN_SUBMODULES ) {
    lock( "lock_testing_device_${DEVICE}" ) {                                    // Монопольный доступ к устройству
        def result = build job: 'target',                                        // Вызов тестирования
                     parameters: [string(name:         'RTOS',                   value: params.RTOS),
                                  string(name:         'SERVER',                 value: params.SERVER),
                                  string(name:         'TARGET',                 value: "$DEVICE"),
                                  string(name:         'TEST_MODULES',           value: "$RUN_MODULES"),
                                  string(name:         'TEST_SUBMODULES',        value: "$RUN_SUBMODULES"),
                                  booleanParam(name:   'UPGRADE',                value: params.UPGRADE),
                                  string(name:         'GIT_BRANCH_JENKINS',     value: params.GIT_BRANCH_JENKINS),
                                  string(name:         'GIT_BRANCH_TESTS',       value: params.GIT_BRANCH_TESTS)]
        copyArtifacts(projectName: 'target',                                     // Сохранение всех артефактов последнего вызова
                      selector:    specific("${result.number}"),
                      target:      "logs_${DEVICE}");
    }
}
```

Тестирование: дочерний конвейер

- Запуск контейнера производится с использованием докер-агента соответствующего релиза

```
pipeline {
    agent {
        docker {
            /* Здесь RTOS и SERVER - переданные от основного конвейера параметры */
            image "kpda-neutrino/${RTOS}:latest"
            label "${SERVER}"
        }
    }
    ...
```

- Сборка тестов осуществляется на отдельном этапе (Stage)

```
stage('Building') {
    steps {
        sh '''
            cd ${WORKSPACE}/target-tests
            make -j16 -Orecurse install 1>${WORKSPACE}/build-tests.log 2>&1            # Сборка с помощью рекурсивной сборочной системы

            cd ${WORKSPACE}
            cp target/test_run.sh ${WORKSPACE}/transfer/opt/testing                    # Копирование скрипта запуска тестов в директорию будущего архива
            cp -r target-tests/.install/neutrino/$CPUVARDIR/*  ${WORKSPACE}/transfer   # Копирование собранных тестов

            tar -C transfer -cpvf ${WORKSPACE}/transfer.tar ./*                        # Создание архива
        '''
    }
}
```

Тестирование: вызов тестов

- Для передачи тестов на устройство и вызова скрипта запуска тестов по SSH используется внутренняя библиотека, обёртка над протоколами SSH/SFTP
- Для вызова тестов используется shell-скрипт, передаваемый на устройство вместе с тестами
- Общая схема его работы:

```
cd ${TESTS_DIR}
...                                                     # Обход иерархии <Модуль>/<Сабмодуль>/<Тест>
while read -r module; do
    cd $module
    while read -r submodule; do
        cd $submodule
        while read -r found_test; do
            cd $found_test

            echo "===Description===" > $found_test.log          # Занесение описания с шапкой вида "===Description===" в лог теста
            cat info.txt >> $found_test.log
            echo "=====Subjects====" >> $found_test.log         # Занесение перечня тестируемых компонентов с шапкой вида "===Subjects===" в лог
            cat subject.txt >> $found_test.log
            echo "================" >> $found_test.log

            ./test &>>$found_test.log                         # Вызов теста с перенаправлением всего вывода в лог
            cp -c ${found_test}.log ${DIR_LOGS}               # Копирование лога теста в общую директорию
            cd ..
        done < ...
        cd ..
    done < ...
    cd ..
done < ...

tar -czf /opt/${ARCHIVE_NAME}.tar.gz logs_${DEVICE}           # Запаковка архива с логами
```

Тестирование: использование expect

- Использование expect на примере одного тест-кейса из теста утилиты login

```
echo -n "Login as root: "
expect -c '''
set timeout 5
spawn login
expect {
    -re "Username:" {                                   # Ожидаемая строка при установленной локали en_EN
        send -- "root\r"                                # Ввод имени пользователя root после приглашения
        exp_continue
    }
    -re "Password:" {
        send -- "12345678\r"                            # Ввод пароля после приглашения
        exp_continue
    }
    -re "Signed in:" {                                  # В успешном сценарии выполняется вход в систему с соответствующим сообщением
        puts "-> Pass"
        exit 0
    }
    timeout {                                           # Иначе выход по истечении времени
        puts "-> Fail"
        exit 1
    }
}
''' &> expect_output
```

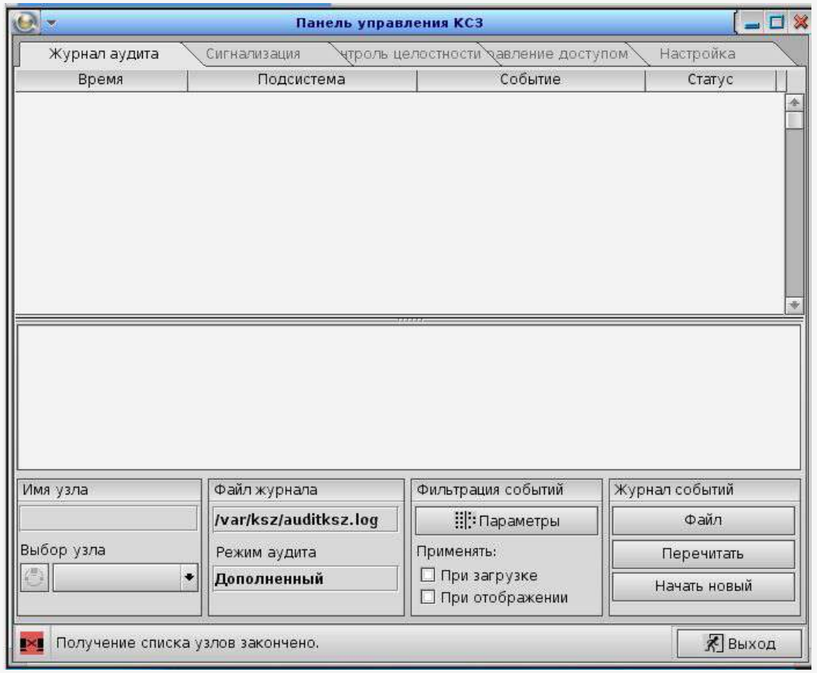
- Содержание expect_output:

```
spawn login
Username: root
Password:
Signed in: 16:47:11 01.10.2024 on /dev/ttyp1
-> Pass
```

Тестирование: виртуальный ввод

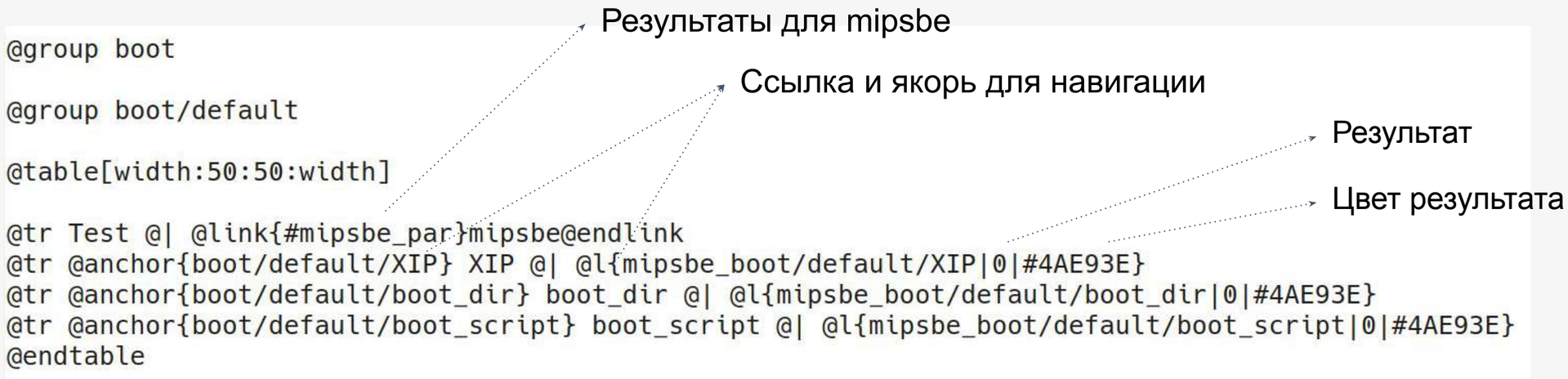
- Использование драйвера виртуального ввода (devi-virtual) на примере завершения приложения phksz по клику на крестик в правом верхнем углу

```
slay_exit() {
    slay -f devi-virtual
    exit $1
}

/usr/photon/bin/devi-virtual vtouch                    # Запуск драйвера виртуального ввода
                                                       # с протоколом vtouch (вирт. тачпада)
echo "Запуск phksz и поиск его в списке процессов"
/usr/photon/bin/phksz 2>&1 &                           # Запуск проверяемого приложения phksz
echo "Поиск процесса в pidin:"
if [ -n "$(pidin | grep phksz)" ]; then                # Поиск запущенного приложения в процессах
    echo "=> Успешно"
else
    echo "=> Неуспешно"
    slay_exit 1
fi

echo "LEFT_BUTTON|690|10" > /dev/virtual-touch         # Передача драйверу команды клика левой клавишей
                                                       # в координатах "крестика" в правом верхнем углу
echo "Повторный поиск процесса в pidin после завершения"
if [ -z "$(pidin | grep phksz)" ]; then                # Поиск в перечне процессов завершенного приложения
    echo "=> Успешно"
else
    echo "=> Неуспешно"
    slay_exit 1
fi

slay_exit 0
```



Тестирование: обработка артефактов

- В конце тестирования из формата внутренней разметки генерируется PDF-отчет
- Пример таблицы с результатами модуля boot:




```
@group boot

@group boot/default

@table[width:50:50:width]

@tr Test @| @link{#mipsbe_par}mipsbe@endlink
@tr @anchor{boot/default/XIP} XIP @| @l{mipsbe_boot/default/XIP|0|#4AE93E}
@tr @anchor{boot/default/boot_dir} boot_dir @| @l{mipsbe_boot/default/boot_dir|0|#4AE93E}
@tr @anchor{boot/default/boot_script} boot_script @| @l{mipsbe_boot/default/boot_script|0|#4AE93E}
@endtable
```

- Заполнение таблицы в скрипте запуска тестов:

```
...
./test &>>$found_test.log
returnCode=$?
found_test_full="${module}/${submodule}/${found_test}"
[ $returnCode -eq 0 ] && color="#4AE93E" || color="#B00000"

echo "@tr @anchor{${found_test_full}} $found_test @| \
    @l{${TARGET}_${found_test_full}|${returnCode}|$color}" >> ${LOGS_DIR}/general
```

Тестирование: публикация артефактов

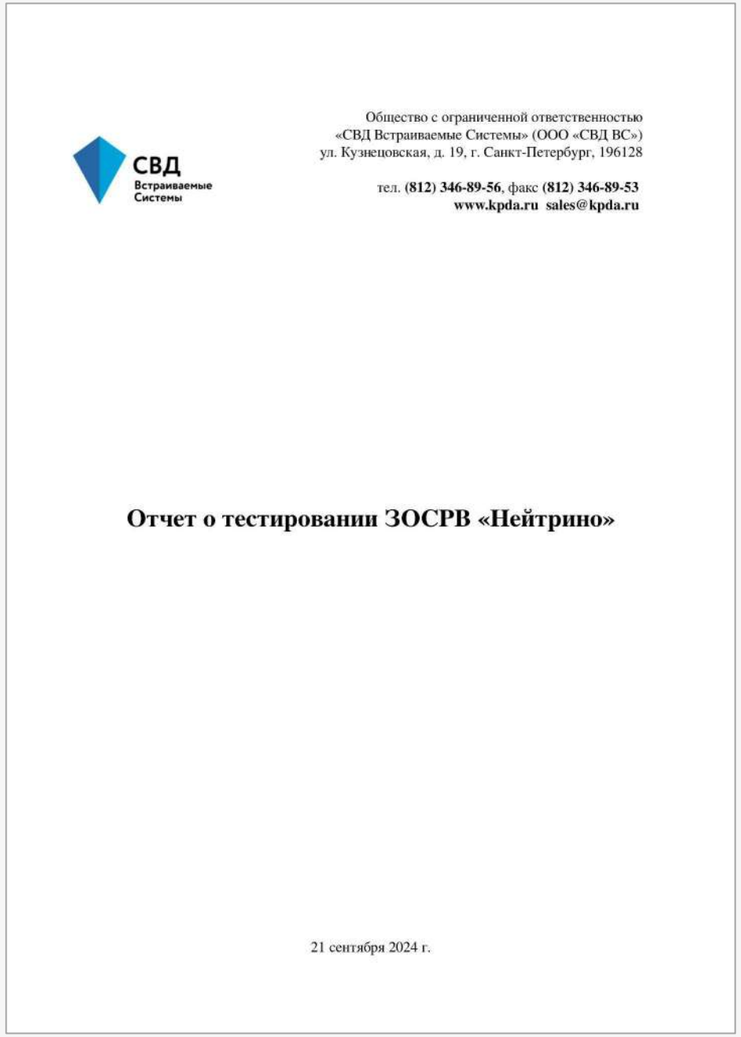
- В конце основного конвейера указывается перечень артефактов к публикации
- Публикация происходит всегда, независимо от результатов
- Опубликованные артефакты не удаляются при следующих запусках

```
post {
    always {                                                                    // Публикация артефактов (всегда, независимо от результата выполнения)
        archiveArtifacts artifacts: "*.log,*.profile,*.svg,*.png,general,*.pdf",
                         allowEmptyArchive: true                                // Не завершать билд со статусом "Failed" при отсутствии артефактов
    }
}
```

- Вид опубликованных артефактов:

Тестирование: пример отчета

Титульный лист



СВД
Встраиваемые Системы

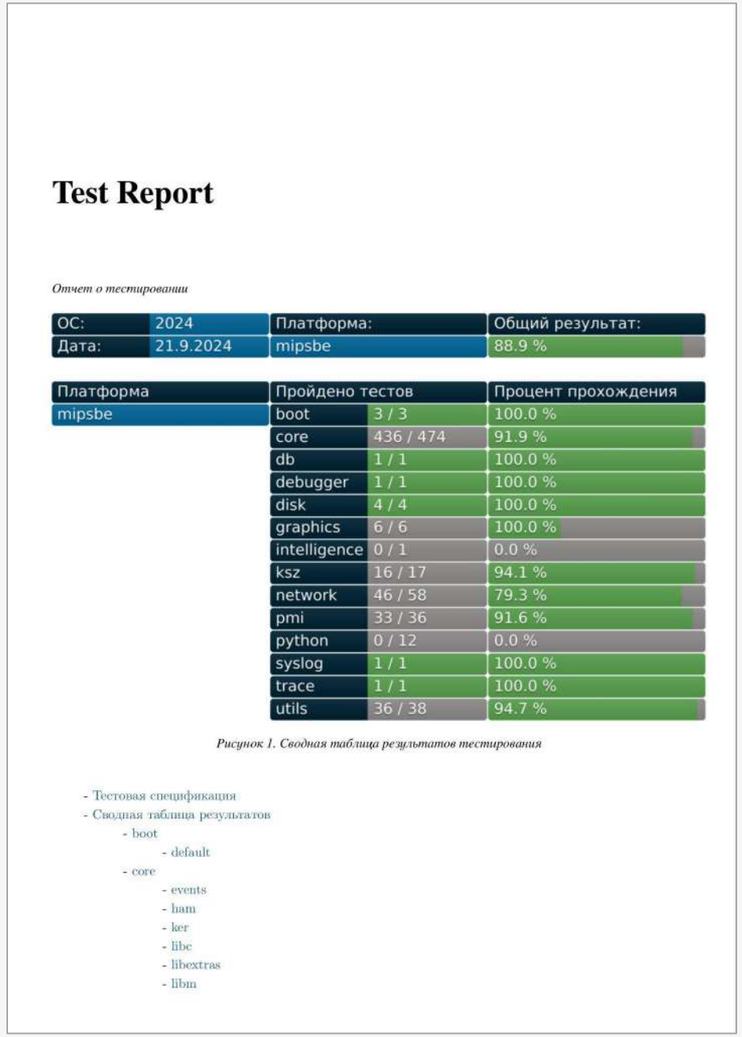
Общество с ограниченной ответственностью
«СВД Встраиваемые Системы» (ООО «СВД ВС»)
ул. Кузнецовская, д. 19, г. Санкт-Петербург, 196128

тел. **(812) 346-89-56**, факс **(812) 346-89-53**
www.kpda.ru sales@kpda.ru

Отчет о тестировании ЗОСРВ «Нейтрино»

21 сентября 2024 г.

Общие результаты



Test Report

Отчет о тестировании

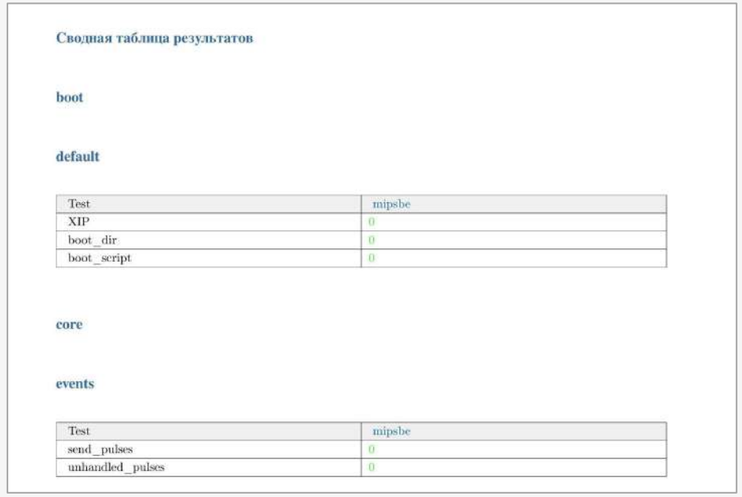
ОС:	2024	Платформа:	Общий результат:
Дата:	21.9.2024	mipsbe	88.9 %

Платформа	Пройдено тестов		Процент прохождения
mipsbe	boot	3 / 3	100.0 %
	core	436 / 474	91.9 %
	db	1 / 1	100.0 %
	debugger	1 / 1	100.0 %
	disk	4 / 4	100.0 %
	graphics	6 / 6	100.0 %
	intelligence	0 / 1	0.0 %
	ksz	16 / 17	94.1 %
	network	46 / 58	79.3 %
	pmi	33 / 36	91.6 %
	python	0 / 12	0.0 %
	syslog	1 / 1	100.0 %
	trace	1 / 1	100.0 %
	utils	36 / 38	94.7 %

Рисунок 1. Сводная таблица результатов тестирования

- Тестовая спецификация
- Сводная таблица результатов
  - boot
    - default
  - core
    - events
    - ham
    - ker
    - libc
    - libextras
    - libm

Результаты и логи тестов



Сводная таблица результатов

boot

default

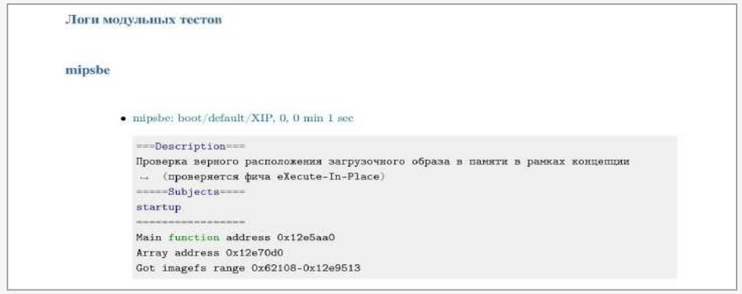
Test	mipsbe
XIP	0
boot_dir	0
boot_script	0

core

events

Test	mipsbe
send_pulses	0
unhandled_pulses	0

…



Логи модульных тестов

mipsbe

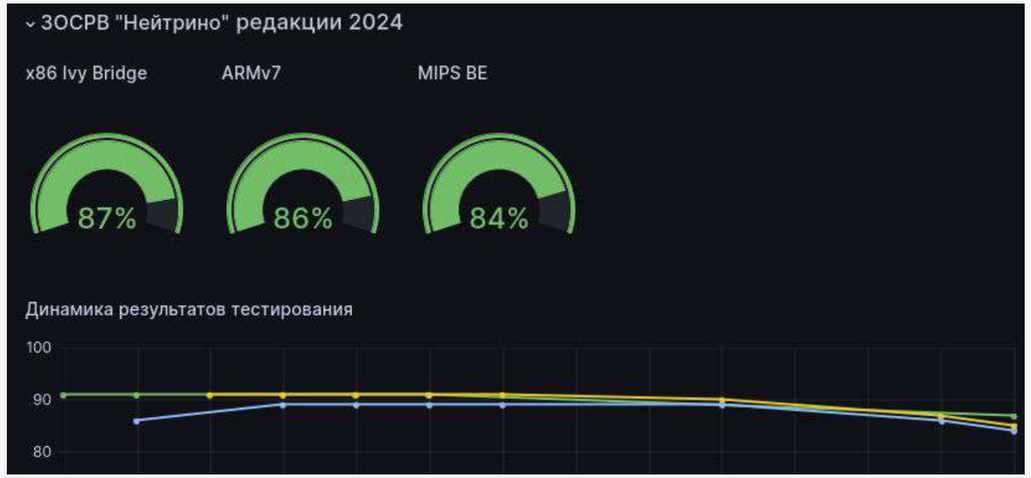
- mipsbe: boot/default/XIP, 0, 0 min 1 sec

```
===Description===
Проверка верного расположения загрузочного образа в памяти в рамках концепции
↪  (проверяется фича eXecute-In-Place)
=====Subjects=====
startup
==================
Main function address 0x12e5aa0
Array address 0x12e70d0
Got imagefs range 0x62108-0x12e9513
```

…

Тестирование: динамика результатов

- Для анализа динамики тестирования используется Grafana, где визуализируются результаты из базы данных



- По результатам анализа кодового покрытия составляется веб-сайт с его демонстрацией

Статистика по компонентам

Компонент	Общий результат	Строки кода	Функции	Ветви условий
/usr/ksz/tests/test_ksz_daccess	81.9%	86.6%	85.7%	73.5%
/usr/ksz/tests/test_ksz_cleanram	78.7%	80.3%	90.9%	64.9%
/usr/ksz/tests/test_ksz_cleandisk	78.0%	73.0%	100.0%	61.1%
/usr/bin/sockstat	75.5%	72.7%	100.0%	53.7%
/usr/ksz/tests/test_ksz_tcpip	70.3%	72.0%	90.0%	48.8%
/usr/ksz/tests/test_ksz_maccess	69.5%	68.4%	95.5%	44.7%
/usr/bin/crontab	68.9%	57.5%	100.0%	49.1%
/sbin/brconfig	68.6%	71.0%	80.0%	54.9%
/usr/bin/passwd-std	68.5%	68.0%	86.4%	51.1%
/usr/bin/passwd-ksz	68.5%	68.0%	86.4%	51.1%
/usr/sbin/if_up	68.3%	58.1%	100.0%	46.7%
/usr/bin/arp	67.1%	66.7%	81.2%	53.4%
/sbin/ifwatchd	65.9%	67.6%	75.0%	55.2%
/usr/libexec/sftp-server	62.9%	59.7%	78.2%	50.8%

Дополнительные материалы

О виртуальном вводе

Обзорная статья

О документации

СВД
Встраиваемые
Системы

Спасибо за внимание!

Анастасия Маштакова

Руководитель группы тестирования
Отдел автоматизации и верификации

ул. Кузнецовская, д. 19,
г. Санкт-Петербург
+7 (812) 346-89-56
www.kpda.ru
support@kpda.ru