ДИАГНОСТИЧЕСКИЕ МЕТОДЫ ОБЕСПЕЧЕНИЯ НАДЕЖНОСТИ И КАЧЕСТВА СЛОЖНЫХ СИСТЕМ

DIAGNOSTIC METHODS FOR ENSURING RELIABILITY AND QUALITY OF COMPLEX SYSTEMS

УДК 004.05
doi:10.21685/2307-4205-2023-2-11

ТЕСТИРОВАНИЕ ДИНАМИЧЕСКОГО ВЫПОЛНЕНИЯ КОДА PYTHON ПРОГРАММ ПРИ ПРОВЕДЕНИИ ЭТАПА СЕРТИФИКАЦИОННЫХ ИСПЫТАНИЙ (РАЗРАБОТКИ) В СИСТЕМЕ СЕРТИФИКАЦИИ МИНОБОРОНЫ РОССИИ

В. В. Самаров

ООО «16 НИИЦ», Мытищи, Московская обл., Россия
samarov_vladimir@mail.ru

Аннотация. *Актуальность и цели.* Во многих случаях программные изделия, подлежащие проверке в системе обязательной сертификации Минобороны России, разрабатываются с помощью интерпретируемого языка программирования Python. При проведении проверок приложений, разработанных на языке Python на предмет соответствия их кода требованиям руководящего документа «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей» (Гостехкомиссия России, М., 1999), далее – РД НДВ, наряду со статическим анализом должен проводиться динамический анализ (для уровней контроля РД НДВ 3 и 2). *Материалы и методы.* Вместе с тем в настоящее время в системе сертификации Минобороны России отсутствуют сертифицированные средства и рекомендованные методы и алгоритмы, которые могут быть использованы для проведения тестирования динамического выполнения кода Python программ на этапе сертификационных испытаний. Перспективным подходом в исследовании недекларированных возможностей кода Python приложений при проведении динамического тестирования является метод, основанный на использовании статистики генерируемой встроенным профайлером интерпретатора Python. *Результаты и выводы.* Подробно описана последовательность действий по формированию с помощью статистических данных, полученных посредством системы профилирования Python, отчетности, анализ которой позволяет сделать обоснованные выводы по соответствию/несоответствию исследуемого кода приложения, разработанного на языке программирования Python требованиям РД НДВ, в части динамического анализа.

Ключевые слова: сертификационные испытания программных изделий, динамический анализ кода Python программ, контроль полноты и отсутствия избыточности на уровне файлов и функциональных объектов, профилирование кода для динамического анализа

Для цитирования: Самаров В. В. Тестирование динамического выполнения кода python программ при проведении этапа сертификационных испытаний (разработки) в системе сертификации Минобороны России // Надежность и качество сложных систем. 2023. № 2. С. 96–103. doi:10.21685/2307-4205-2023-2-11

TESTING THE DYNAMIC EXECUTION OF PYTHON PROGRAM CODE DURING THE CERTIFICATION TESTING (DEVELOPMENT) STAGE IN THE CERTIFICATION SYSTEM OF THE MINISTRY OF DEFENSE OF RUSSIA

V.V. Samarov

LLC "16 NIITS", Mytishchi, Moscow region, Russia
samarov_vladimir@mail.ru

Abstract. *Background.* In many cases, software products subject to verification in the mandatory certification system of the Russian Ministry of Defense are developed using the interpreted Python programming language. When checking applications developed in the Python language for compliance of their code with the requirements of the governing document "Protection against unauthorized access to information. Part 1. Information security software. Classification according to the level of control of the absence of undeclared capabilities" (State Technical Commission of Russia, Moscow, 1999) 1, hereinafter – RD NDV, along with static analysis, a dynamic analysis should be carried out (for control levels of RD NDV 3 and 2). *Materials and methods.* At the same time, at present, the certification system of the Russian Ministry of Defense lacks certified tools and recommended methods and algorithms that can be used to test the dynamic execution of the Python code of programs at the stage of certification tests. A promising approach in the study of undeclared capabilities of the Python code of applications during dynamic testing is a method based on the use of statistics generated by the built-in profiler of the Python interpreter. *Results and conclusions.* The article describes in detail the sequence of actions for generating, using statistical data obtained through the Python profiling system, reporting, the analysis of which allows drawing reasonable conclusions on the compliance / non-compliance of the studied application code developed in the Python programming language with the requirements of the RD NDV, in terms of dynamic analysis.

Keywords: certification tests of software products, dynamic code analysis of Python programs, control of completeness and lack of redundancy at the level of files and functional objects, code profiling for dynamic analysis

For citation: Samarov V.V. Testing the dynamic execution of python program code during the certification testing (development) stage in the certification system of the ministry of defense of Russia. *Nadezhnost' i kachestvo slozhnykh sistem = Reliability and quality of complex systems.* 2023;(2):96–103. (In Russ.). doi:10.21685/2307-4205-2023-2-11

В системе сертификации Минобороны России динамический анализ исследуемого кода проводится в соответствии с требованиями РД НДВ [1].

В соответствии с этим документом при проведении динамического анализа должны быть осуществлены:

– контроль выполнения функциональных объектов (ФО);

– сопоставление фактических маршрутов выполнения функциональных объектов и маршрутов, построенных в процессе проведения статического анализа.

Для выполнения перечисленных требований при анализе проектов, разработанных на языке Python, предлагается воспользоваться статистикой, которая может быть получена профилировщиком интерпретатора Python – cProfile [2].

Действительно, ранее автором была рассмотрена отчетность, предоставляемая профилировщиком cProfile, и установлено, что генерируемой в результате профилирования статистической информации достаточно для формирования обоснованного вывода как по отработавшим функциональным объектам, так и по маршрутам их выполнения [3].

Для тестирования динамического выполнения кода Python программ при проведении сертификационных испытаний (этапа разработки) с использованием профилировщика cProfile (системы профилирования Python) предлагается воспользоваться следующим алгоритмом (на примере тестирования кросс-платформенного проекта, разработанного на языке Python, функционирующего в том числе в сертифицированной ОС «Astra Linux Special Edition» 1.6) [4]:

1. С помощью встроенного профилировщика cProfile получить статистику вызовов ФО исследуемого проекта при его выполнении в формате pstats [2], для чего в терминале выполнить команду:

python –m cProfile –o proj_name.pstats proj_main_module_app.py,

где *proj_name.pstats* – результаты профилирования проекта *proj_name* с главным модулем *proj_main_module_app.py* в потоковом формате pstats.

После чего произвести отработку заявленных функциональных возможностей исследуемого проекта. По окончании отработки функциональных возможностей исследуемого проекта зафиксировать сформированный файл *proj_name.pstats proj.*

2. Получить из файла *proj_name.pstats* статистическую информацию в текстовом виде, для чего в терминале выполнить команду:

python -c "import pstats; p = pstats.Stats('proj_name.pstats'); p.sort_stats('time').print_stats()" > proj_name_prof.txt

После чего зафиксировать полученный текстовый файл *proj_name_prof.txt*

3. С помощью питон-модуля *gprof2dot* [5] и результатов профилирования, сформированных при выполнении п.1, получить граф (файл *proj_name.png* *) с деревом фактических вызовов функций исследуемого проекта, для чего в терминале выполнить следующие команды:

pip3 list // просмотр установленных в системе пакетов

pip3 install gprof2dot // установка питон библиотеки *gprof2dot* (в случае отсутствия данного пакета в системе)

sudo apt list graphviz *// получение информации о пакете *graphviz* (установлен/не установлен)

sudo apt install graphviz *// установка пакета *graphviz* (если не установлен)

gprof2dot –f pstats proj_name.pstats | dot –Tpng –o proj_name.png

После чего зафиксировать полученный графический файл *proj_name.png*

Примечание: для построения графов, в системе должно быть установлено ПО «Graphviz» [6] (входит в состав ОС «Astra Linux Special Edition» 1.6. В случае проведении испытаний в ОС семейства «Windows» ПО «Graphviz» должно быть установлено дополнительно).

4. Произвести формирование текстового файла *FO_proj_name.txt* с перечнем функциональных объектов в виде

Path_to_analized_proj+ file_name1_py[Num_Str]FO_name[1]+(«.fnc»)
Path_to_analized_proj+ file_name1_py[Num_Str]FO_name[2]+(«.fnc»)
……
Path_to_analized_proj+ file_name1_py[Num_Str]FO_name[A]+(«.fnc»)
Path_to_analized_proj+ file_name2_py[Num_Str]FO_name[1]+(«.fnc»)
……
Path_to_analized_proj+ file_nameN_py[Num_Str]FO_name[Z]+(«.fnc»)

(где:

Path_to_analized_proj – абсолютный путь к каталогу с анализируемым проектом;

file_name_py – имя файла из каталога с анализируемым проектом;)

file_name_py[Num_Str]FO_name – имя ФО (класса, функции) из файла file_name_py (ФО объявлен в строке [Num_Str] файла);

«.fnc» – технологический маркер, обозначающий класс/функцию как объект, для которого применимы операции, предоставляемые файловой системой.

Примечание: для получения статистических данных, необходимых для дальнейшего получения перечня функциональных объектов в представленном виде, требуется воспользоваться существующими инструментами статического анализа (например: утилитой ctags [7], статическим анализатором исходных текстов – «SCI Understand» [8] и др.) или разработать Python утилиту, осуществляющую построение абстрактного синтаксического дерева (использование стандартного Python-модуль «ast» [9] позволяет сформировать перечень функциональных объектов анализируемого проекта (в том числе с корректной обработкой функций-декораторов, а также асинхронных функций).

5. По исходным тестам исследуемого проекта сформировать граф возможных вызовов ФО, для чего в терминале выполнить следующие действия:

pip3 list // просмотр установленных в системе пакетов

pip3 install code2flow // установка питон библиотеки *code2flow*

*code2flow –language = py Path_to_analized_proj –o proj_name_stat.png **

После этого зафиксировать полученный графический файл *proj_name_stat.png,* представляющий собой граф вызовов функциональных объектов исследуемого проекта.

Примечание: для построения графов в системе должно быть установлено ПО «Graphviz» (входит в состав ОС «Astra Linux». В случае проведения испытаний в ОС «Windows» ПО «Graphviz» должно быть установлено дополнительно).

6. Осуществить преобразование данных, полученных на этапе выполнения п. 2 (файл *proj_name_prof.txt*), к представлению, аналогично описанному в п.4.

7. Произвести сравнение перечня функциональных объектов, сформированного по результатам статического анализа (п. 4), с фактически отработавшими ФО (п. 6), для этого выполнить следующие действия:

1) скопировать в каталог *\data* «ПИ «ПМАКФИ-16» [10] файлы *FO_proj_name.txt* и *proj_name_prof.txt*;

2) запустить ПИ «ПМАКФИ-16» с параметрами:

python app.py --first = data/ FO_proj_name.txt --second = data/proj_name_prof.txt

--dest_path = PROJ_DIR --reg = 1 --exts = "fnc"

где *app.py* – главный модуль ПИ «ПМАКФИ-16»; *--first* – параметр с указанием пути к первому файлу *FO_proj_name.txt,* сформированному на этапе выполнения п. 4; *--second* – параметр с указанием пути ко второму файлу *proj_name_prof.txt,* сформированному на этапе выполнения п. 2; *--dest_path = PROJ_DIR* – имя каталога, по которому производится анализ (основной каталог запуска питон проекта).

8. Перейти в каталог *\result* «ПИ «ПМАКФИ-16». Ознакомиться с файлами отчетов (*Rep_equals.txt* – перечень ФО, определенных в результате статического анализа и фактически отработавших по результатам динамического анализа; *Rep_file1_only.txt* – перечень ФО, определенных в результате статического анализа, но отработавших по результатам динамического анализа; *Rep_file2_only.txt** – перечень ФО, фактически отработавших по результатам динамического анализа, но отсутствующих в файлах исходных текстов исследуемого проекта по результатам статического анализа (рис. 1); *result.html* – сводный html отчет с результатами анализа, рис. 2).

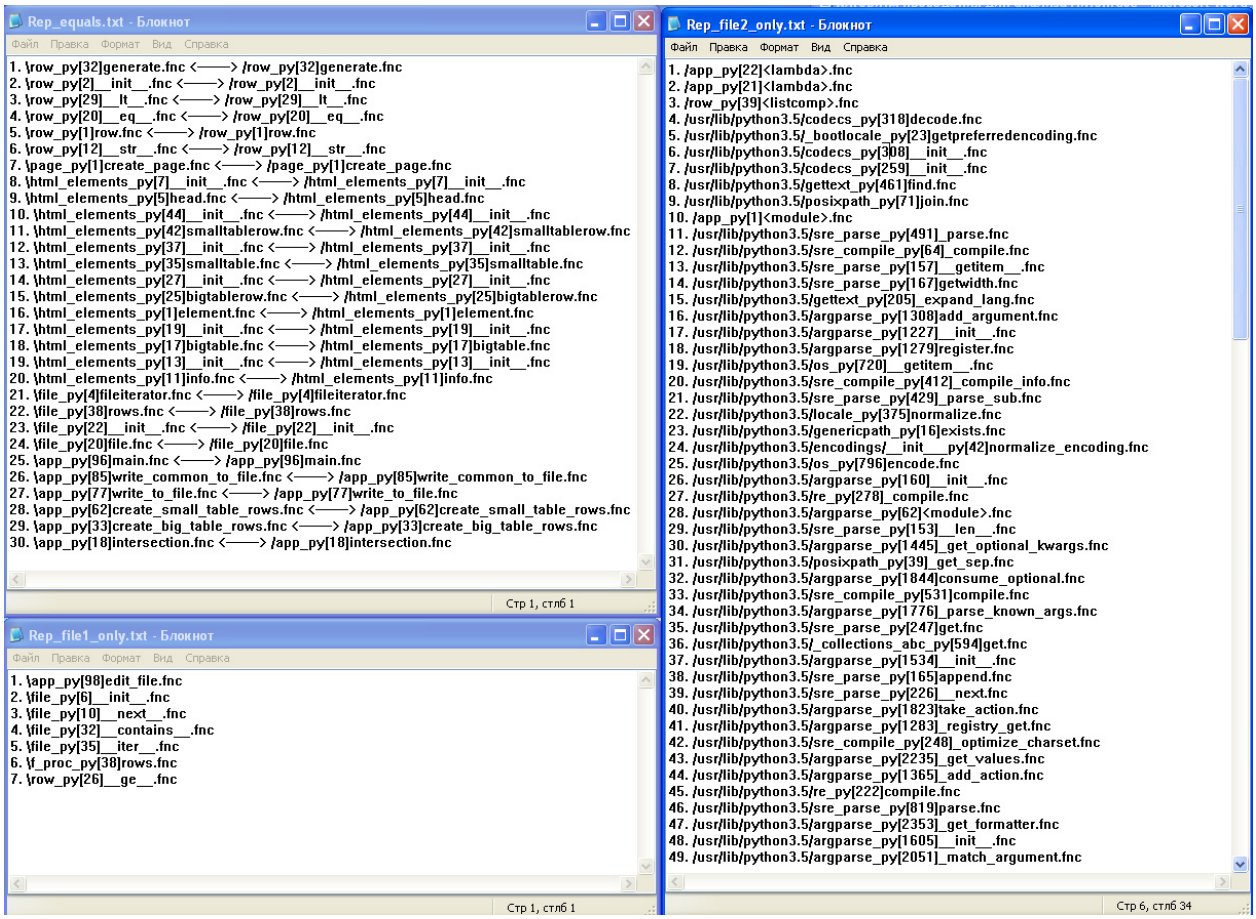
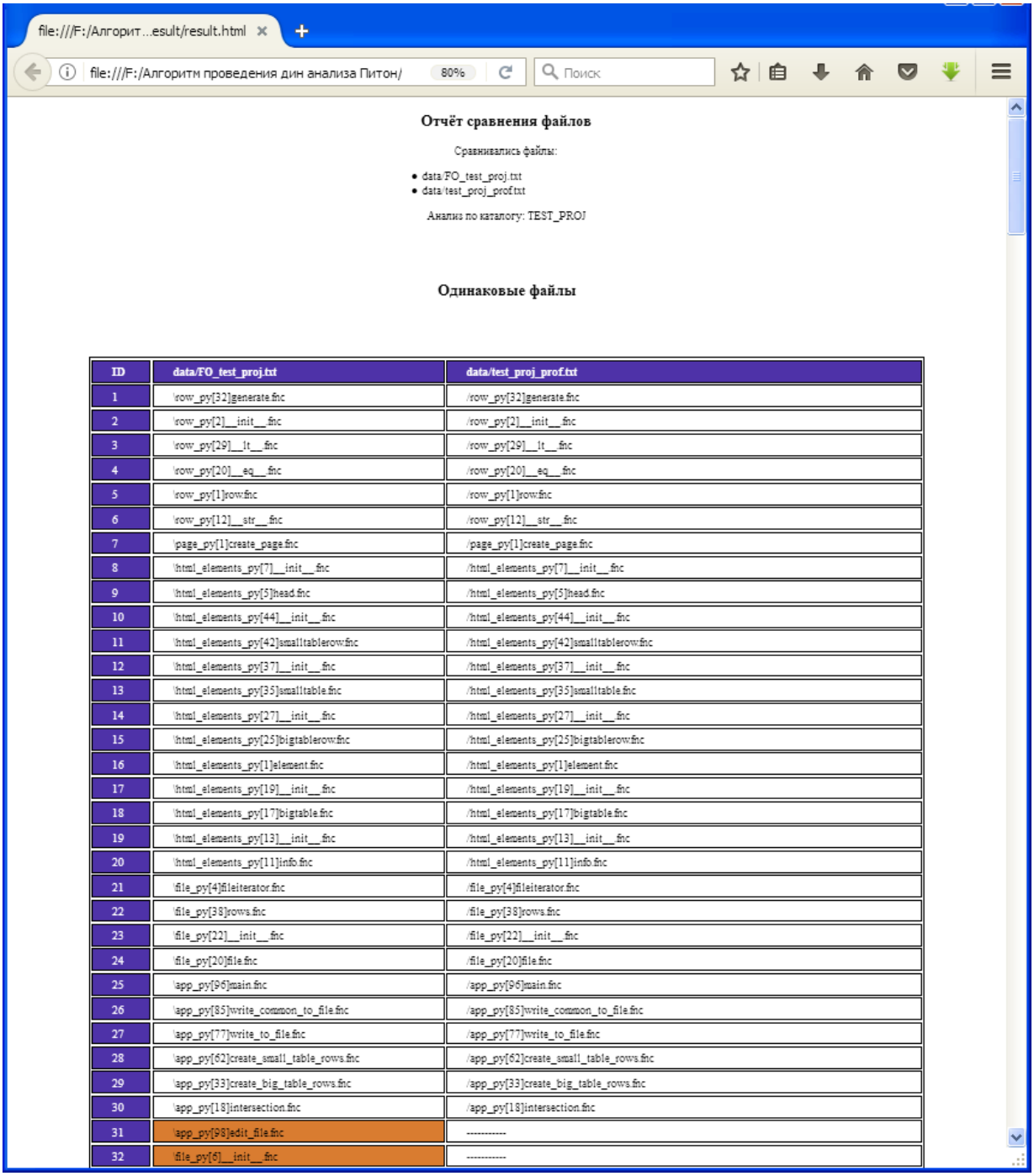


Рис. 1. Видеокадр отчетов *Rep_equals.txt*, *Rep_file1_only.txt* и *Rep_file2_only.txt*, сформированных по результатам анализа тестового проекта



ID	data/FO_test_proj.txt	data/test_proj_prof.txt
1	\row_py[32]generate.fnc	/row_py[32]generate.fnc
2	\row_py[2]__init__.fnc	/row_py[2]__init__.fnc
3	\row_py[29]__lt__.fnc	/row_py[29]__lt__.fnc
4	\row_py[20]__eq__.fnc	/row_py[20]__eq__.fnc
5	\row_py[1]row.fnc	/row_py[1]row.fnc
6	\row_py[12]__str__.fnc	/row_py[12]__str__.fnc
7	\page_py[1]create_page.fnc	/page_py[1]create_page.fnc
8	\html_elements_py[7]__init__.fnc	/html_elements_py[7]__init__.fnc
9	\html_elements_py[5]head.fnc	/html_elements_py[5]head.fnc
10	\html_elements_py[44]__init__.fnc	/html_elements_py[44]__init__.fnc
11	\html_elements_py[42]smalltablerow.fnc	/html_elements_py[42]smalltablerow.fnc
12	\html_elements_py[37]__init__.fnc	/html_elements_py[37]__init__.fnc
13	\html_elements_py[35]smalltable.fnc	/html_elements_py[35]smalltable.fnc
14	\html_elements_py[27]__init__.fnc	/html_elements_py[27]__init__.fnc
15	\html_elements_py[25]bigtablerow.fnc	/html_elements_py[25]bigtablerow.fnc
16	\html_elements_py[1]element.fnc	/html_elements_py[1]element.fnc
17	\html_elements_py[19]__init__.fnc	/html_elements_py[19]__init__.fnc
18	\html_elements_py[17]bigtable.fnc	/html_elements_py[17]bigtable.fnc
19	\html_elements_py[13]__init__.fnc	/html_elements_py[13]__init__.fnc
20	\html_elements_py[11]info.fnc	/html_elements_py[11]info.fnc
21	\file_py[4]fileiterator.fnc	/file_py[4]fileiterator.fnc
22	\file_py[38]rows.fnc	/file_py[38]rows.fnc
23	\file_py[22]__init__.fnc	/file_py[22]__init__.fnc
24	\file_py[20]file.fnc	/file_py[20]file.fnc
25	\app_py[96]main.fnc	/app_py[96]main.fnc
26	\app_py[85]write_common_to_file.fnc	/app_py[85]write_common_to_file.fnc
27	\app_py[77]write_to_file.fnc	/app_py[77]write_to_file.fnc
28	\app_py[62]create_small_table_rows.fnc	/app_py[62]create_small_table_rows.fnc
29	\app_py[33]create_big_table_rows.fnc	/app_py[33]create_big_table_rows.fnc
30	\app_py[18]intersection.fnc	/app_py[18]intersection.fnc
31	\app_py[98]edit_file.fnc	----------
32	\file_py[6]__init__.fnc	----------

Рис. 2. Видеокадр сводного отчета *result.html*, сформированного по результатам анализа тестового проекта

9. По результатам анализа отчетности, сформированной «ПИ «ПМАКФИ-16», сделать заключения:

– о фактической полноте отработки функциональных объектов из состава пакета с исходными текстами исследуемого проекта (файлы *Rep_equals.txt* и *Rep_file1_only.txt*) и, как следствие, отсутствие (наличие) функциональной и файловой избыточности;

– по полноте исходных текстов на уровне файлов и на уровне функциональных объектов.

Примечание: при анализе файла отчета *Rep_file2_only.txt* обратить внимание на файлы, из которых вызывались функциональные объекты, с целью определения источника и статуса соответствующего программного модуля. Статус программного модуля может быть:

– верифицированный, в случае, если данный программный модуль получен из надежного источника (например, из состава интерпретатора Python версии 3.5, входящего в состав сертифицированной ОС «Astra Linux» 1.6);

– неверифицированный, в случае, если источник получения данного программного модуля (библиотеки) не установлен/сомнительный (при идентификации статуса используемых файлов особое внимание уделить Python файлам из каталогов «Site-Packages» и «Dist-packages», а также версии фактически используемого интерпретатора Python).

10. Ознакомиться с файлом *proj_name.png,* представляющим собой граф фактических вызовов функций тестового проекта (сформирован на этапе выполнения действий описанных в п. 3, рис. 3 (справа), сравнить его с графом вызовов функций (файл *proj_name_stat.png,* сформирован на этапе выполнения п. 5), построенных в результате анализа исходных текстов проекта, рис. 3 (слева)).

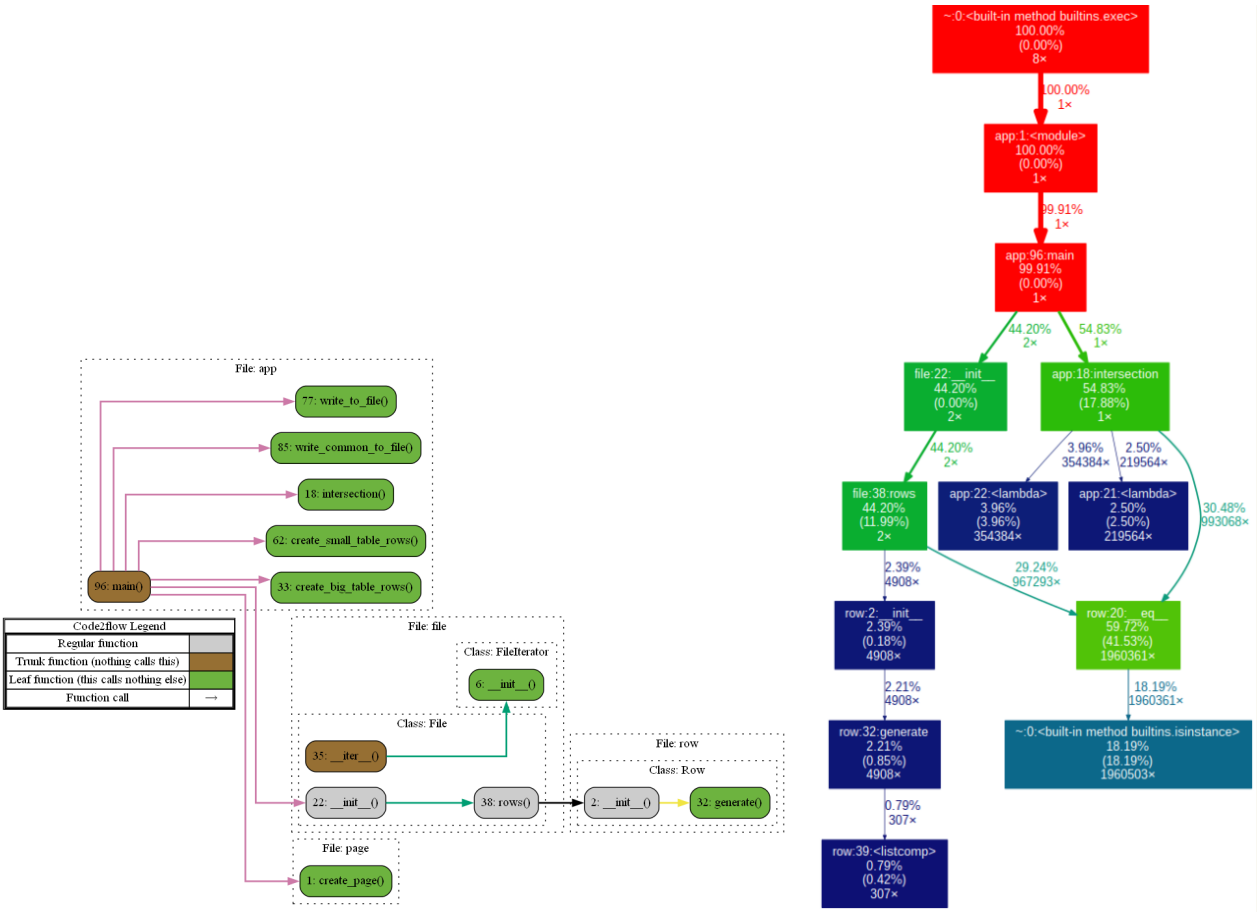


Рис. 3. Видеокадр файлов отчетов, представляющих собой графы вызовов функциональных объектов тестового проекта, сформированных по исходным текстам на этапе статического анализа (слева) и по результатам фактического исполнения кода программы (справа)

11. Сделать заключение о соответствии (несоответствии) фактических маршрутов выполнения функциональных объектов, сформированных по результатам выполнения исследуемого проекта, с возможными маршрутами выполнения ФО, сформированными по результатам выполнения статического анализа (при анализе акцентировать внимание на обязательность выполнения критических маршрутов выполнения программы).

Заключение

Приведенный алгоритм, реализующий исследовательский подход, основанный на использовании системы профилирования cProfile, может быть использован специалистами испытательных лабораторий (аккредитованных в системах сертификации Минобороны России) при проведении соответствующих этапов сертификационных испытаний, а также разработчиками программных изделий (предприятиями промышленности) на этапе разработки и тестирования.

Список литературы

1. Защита от несанкционированного доступа к информации Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей. URL: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty/114-spetsialnye-normativnye-dokumenty/382-rukovodyashchij-dokument-prikaz-predsedatelya-gostekhkomissii-rossii-ot-4-iyunya-1999-g-n-114 (дата обращения: 20.02.2023).
2. 27.4. The Python Profilers – Python 3.5.9 documentation. URL: https://docs.python.org/3.5/library/profile.html (дата обращения: 20.02.2023).
3. Старостин И. Е. Программная реализация решения потенциально-потоковым методом задач построения моделей систем из результатов испытаний этих систем // Надежность и качество сложных систем. 2020. № 3. С. 128–136. doi:10.21685/2307-4205-2020-3-15
4. Операционная система Astra Linux Special Edition: описание, назначение, применение. URL: https://astralinux.ru/products/astra-linux-special-edition (дата обращения: 20.02.2023).
5. gprof2dot ‘ PyPi. URL: https://pypi.org/project/gprof2dot (дата обращения: 20.02.2023).
6. Graphviz. URL: https://graphviz.org (дата обращения: 20.02.2023).
7. Universal Ctags. URL: https://ctags.io (дата обращения: 20.02.2023).
8. Understand: An IDE and Static Code analysis Tool by SciTools. URL: https://www.scitools.com (дата обращения: 20.02.2023).
9. 32.2. ast – Abstract Syntax Trees – Python 3.5.9 documentation. URL: https://docs.python.org/3.5/library/ast.html (дата обращения: 20.02.2023).
10. Самаров В. В., Юрков Н. К. Программная реализация контроля файловой избыточности и подтверждения полноты исходных текстов на уровне файлов // Надежность и качество сложных систем. 2021. № 2. С. 104–108. doi:10.21685/2307-4205-2021-2-11

References

1. *Zashchita ot nesanktsionirovannogo dostupa k informatsii Chast' 1. Programmnoe obespechenie sredstv zashchity informatsii. Klassifikatsiya po urovnyu kontrolya otsutstviya nedeklarirovannykh vozmozhnostey = Protection against unauthorized access to information Part 1. Information security software. Classification according to the level of control of the absence of undeclared opportunities*. (In Russ.). Available at: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty/114-spetsialnye-normativnye-dokumenty/382-rukovodyashchij-dokument-prikaz-predsedatelya-gostekhkomissii-rossii-ot-4-iyunya-1999-g-n-114 (accessed 20.02.2023).
2. *27.4. The Python Profilers – Python 3.5.9 documentation*. Available at: https://docs.python.org/3.5/library/profile.html (accessed 20.02.2023).
3. Starostin I.E. Software implementation of the solution of problems of constructing models of systems from the test results of these systems by a potentially streaming method. *Nadezhnost' i kachestvo slozhnykh system = Reliability and quality of complex systems*. 2020;(3):128–136. (In Russ.). doi:10.21685/2307-4205-2020-3-15
4. *Operatsionnaya sistema Astra Linux Special Edition: opisanie, naznachenie, primenenie = Astra Linux Special Edition operating system: description, purpose, application*. (In Russ.). Available at: https://astralinux.ru/products/astra-linux-special-edition (accessed 20.02.2023).
5. *gprof2dot ‘ PyPi*. Available at: https://pypi.org/project/gprof2dot (accessed 20.02.2023).
6. *Graphviz*. Available at: https://graphviz.org (accessed 20.02.2023).
7. *Universal Ctags*. Available at: https://ctags.io (accessed 20.02.2023).
8. *Understand: An IDE and Static Code analysis Tool by SciTools*. Available at: https://www.scitools.com (accessed 20.02.2023).
9. *32.2. ast – Abstract Syntax Trees – Python 3.5.9 documentation*. Available at: https://docs.python.org/3.5/library/ast.html (accessed 20.02.2023).
10. Samarov V.V., Yurkov N.K. Software implementation of file redundancy control and confirmation of completeness of source texts at the file level. *Nadezhnost' i kachestvo slozhnykh sistem = Reliability and quality of complex systems*. 2021;(2):104–108. (In Russ.). doi:10.21685/2307-4205-2021-2-11

Информация об авторах / Information about the authors

Владимир Владимирович Самаров
заместитель начальника испытательной лаборатории,
ООО «16 НИИЦ»
(Россия, Московская обл., г. Мытищи,
Олимпийский пр-т, 29)
E-mail: samarov_vladimir@mail.ru

Vladimir V. Samarov
Deputy Head of the testing laboratory,
LLC "16 NIITS"
(29 Olimpiyskiy avenue, Mytishchi,
Moscow region, Russia)

Автор заявляет об отсутствии конфликта интересов /
The author declares no conflicts of interests.

Поступила в редакцию/Received 20.02.2023
Поступила после рецензирования/Revised 10.03.2023
Принята к публикации/Accepted 25.03.2023