МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут
комп'ютерних наук та управління проектами
(повне найменування інституту, назва факультету)

Програмного забезпечення автоматизованих систем
(повна назва кафедри)

Пояснювальна записка

до кваліфікаційної (магістерської) роботи

за темою «Побудова моделі хмарного тестування програмного забезпечення 1С та розробка програми для її реалізації»

Виконав: студент 6 курсу, групи 6151м
спеціальності
121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)
Грущинський О. О
(підпис, прізвище та ініціали)
Керівник Фаріонова Т.А.
(підпис, прізвище та ініціали)
Рецензент Приходько С.Б.
(підпис, прізвище та ініціали)
Завідувач кафедри Приходько С.Б.
(підпис, прізвище та ініціали)

м. Миколаїв – 2020 р.

Міністерство освіти і науки України
Національний університет кораблебудування
імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра програмного забезпечення автоматизованих систем
Освітній ступень Магістр
Галузь 12 «Інформаційні технології»
(шифр і назва)
Спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

" 26 " 10 2020 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ (МАГІСТЕРСЬКУ) РОБОТУ СТУДЕНТУ

Грущинському Олегу Олександровичу
(прізвище, ім'я, по батькові)

1.Тема магістерської роботи «Побудова моделі хмарного тестування програмного забезпечення 1С та розробка програми для її реалізації»
керівник роботи к.т.н. доц. Фаріонова Т.А.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 26 " жовтня 2020 року №1037-уч
2. Строк подання студентом роботи 01.12.2020 року
3. Вихідні дані до роботи
4.Зміст магістерської роботи (МР):
• Титульний аркуш, завдання на кваліфікаційну (магістерську) роботу, реферат (українською, англійською), зміст, перелік умовних позначень, символів, одиниць та термінів (за необхідності).
• Вступ (Актуальність теми. Зв'язок роботи з науковими програмами, планами, темами. Мета і завдання дослідження. Об'єкт дослідження. Предмет дослідження. Методи дослідження. Наукова новизна одержаних результатів. Практичне значення одержаних результатів. Особистий внесок здобувача. Апробація результатів досліджень**.** Публікації.)
• Огляд літератури за темою, обґрунтування необхідності проведення досліджень за обраною темою, вибір напрямків досліджень, мета дослідження, основні задачі дослідження
• Викладення результатів власних досліджень з висвітленням того нового, що пропонується
• Проект програмного забезпечення
• Результати досліджень та розробки проекту програмного забезпечення
• Організаційно-економічний розділ
• Розділи з охорони праці та охорони навколишнього середовища
• Висновки
• Список використаних джерел
• Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма і методика випробувань програмного забезпечення)

5. Перелік графічного матеріалу

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

Назва етапів кваліфікаційної (магістерської) роботи (МР)	Термін виконання	Примітка
1. Підготовка вступної частини МР	18.10.2020	виконано
2. Підготовка розділу (ів) МР з огляду літератури за темою, обґрунтування необхідності проведення досліджень за обраною темою, вибір напрямів досліджень	19.10.2020	виконано
3. Підготовка розділу (ів) МР з результатів власних досліджень	22.10.2020	виконано
4. Підготовка розділу МР з проекту програмного забезпечення	16.11.2020	виконано
5. Підготовка організаційно-економічного розділу	18.11.2020	виконано
6. Підготовка розділу з охорони праці	20.11.2020	виконано
7. Підготовка розділу з охорони навколишнього середовища	23.11.2020	виконано
8. Підготовка розділу МР – Висновки	25.11.2020	виконано
9. Оформлення списку використаних джерел	27.11.2020	виконано
10. Оформлення додатків	30.11.2020	виконано
11. Підготовка презентації МР та доповіді	01.12.2020	виконано
12. Подання МР на попередній захист	01.12.2020	виконано
13. Подання МР рецензенту	11.12.2020	виконано
14. Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, разом з заявою щодо самостійності виконання роботи та ідентичності друкованої та електронної версії роботи	14.12.2020	виконано
15. Подання на кафедру ПЗАС електронних версії наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК від 19.05.2020 р. за №287-уч); презентації доповіді	18.12.2020	виконано
16. Подання на кафедру ПЗАС письмового відгуку та рецензії на кваліфікаційну роботу	18.12.2020	виконано

Студент _____ Грущинський О.О. _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ Фаріонова Т.А._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Грущинський Олег Олександрович

«Побудова моделі хмарного тестування програмного забезпечення 1С та розробка програми для її реалізації»

Кваліфікаційна робота на здобуття освітнього рівня магістра зі спеціальності 121 – «Інженерія програмного забезпечення». Національний університет кораблебудування імені адмірала Макарова. Миколаїв, 2020 р.

Обсяг роботи: 146 стор., 9 табл., 27 рис., 32 використаних джерел, 5 додатків.

Актуальність теми роботи: Важливою задачею розробників ПЗ є впорядкування дій з тестування програмних продуктів 1С: Підприємство з метою раціонального розподілу ресурсів, прийняття обґрунтованих рішень щодо початку та завершення відповідного процесу. Хмарне тестування, допомагає забезпечити потребу вендоров ПЗ в апаратних і програмних ресурсах за рахунок застосування гнучкої і відносно недорогої хмарної платформи для тестування. Таким чином, побудова ефективної моделі хмарного тестування застосунків 1С: Підприємство та розробка відповідного ПЗ є актуальною задачею та представляє науково-практичний інтерес.

Мета та завдання дослідження. Метою є підвищення ефективності тестування ПЗ, яке створене із використанням технологічної платформи 1С, шляхом розробки моделі хмарного тестування, що надає користувачеві можливість скоротити витрати в порівнянні із традиційними методами тестування.

Завдання дослідження: проаналізувати сучасні технології хмарного тестування ПЗ та методологічні підходи до розробки моделей хмарного тестування ПЗ; розробити модель хмарного тестування ПЗ застосунків 1С та оцінити її ефективність; розробити програму для реалізації моделі хмарного тестування програмного забезпечення 1С.

Об'єктом дослідження є процеси тестування програмного забезпечення.

Предметом дослідження є моделі хмарного тестування програмного забезпечення 1С.

Методи дослідження: об’єктно-орієнтованого аналізу та проектування ПЗ, хмарної обробки інформації та тестування.

Наукова новизна одержаних результатів. Розроблено модель хмарного тестування ПЗ 1С, яка в порівнянні з традиційними методами тестування програмних продуктів забезпечує скорочення витрат на проектування та розробку ПЗ.

Практичне значення одержаних результатів. Розроблено програму для хмарного тестування застосунків 1С: Підприємство та 1С:Бітрикс

Ключові слова: ХМАРНІ ОБЧИСЛЕННЯ, ХМАРНЕ ТЕСТУВАННЯ, ХМАРНА ПЛАТФОРМА, ДОДАТОК 1С:ПІДПРИЄМСТВО

ABSTRACT

Grushchinsky Oleg Alexandrovich

«Construction the model of cloud software testing providing 1C and developing a program for its implementation»

The qualification work in obtaining a master's degree in specialty 121 - "Software Engineering". Admiral Makarov National University of Shipbuilding. Mykolayiv, 2020.

The qualification work is presented on the 147 pages of typewritten text, contains 9 tables, 27 figures, 5 appendices and 32 references.

Relevance of the topic of the work. An important task of software developers is to streamline the testing of software products 1C: Enterprise in order to rationally allocate resources, make informed decisions about the beginning and end of the process. Cloud testing helps to meet the need of software vendors for hardware and software resources by the use of a flexible and relatively inexpensive cloud platform for testing. Thus, the construction of an effective model of cloud testing of 1C: Enterprise applications and the development of appropriate software is an urgent task and is of scientific and practical interest.

The goal and objectives of the study. The goal of the study is to increase the efficiency of software testing, which is created using the technological platform 1C, by developing a model of cloud testing, which gives the user the opportunity to reduce costs compared to traditional testing methods.. The objectives of the study: to analyze modern technologies of cloud software testing and methodological approaches to the development of cloud software testing models; develop a model of cloud testing of 1C applications software and evaluate its effectiveness; to develop a program for the implementation of the model of cloud testing software 1C.

The object of the study is the process of software testing.

The subject of the study is the model of cloud testing of 1C software/

The methods of the study: the object-oriented analysis and design of software, cloud information processing and testing.

The scientific novelty of obtained results. A model of cloud testing of 1C software has been developed, which in comparison with traditional methods of software testing provides reduction of software design and development costs.

The practical significance of obtained results. A program for cloud testing of 1C: Enterprise and 1C: Bitrix applications has been developed

Keywords: CLOUD COMPUTING, CLOUD TESTING, CLOUD PLATFORM, APPLICATION 1C: ENTERPRISE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ІТ – інформаційна технологія

ПЗ – програмне забезпечення

AWS – Amazon Web Services

ER – Entity-Relationship

GCP – Google Cloud Platform

IaaS – інфраструктура як послуга

ISO – International System Organization

PaaS – платформа як послуга

SaaS – програмне забезпечення як послуга

TaaS – тестування як послуга

UML – Unified Modelling Language

ВСТУП

Важливою задачею розробників програмного забезпечення є впорядкування дій з тестування у вигляді зв’язаного процесу з метою раціонального розподілу ресурсів, а також прийняття обґрунтованих рішень щодо початку та завершення тестування програмного забезпечення. Це пов’язано із необхідністю скорочення витрат на розробку програного продукту шляхом обмеження фінансових і трудових витрат на виконання повноцінного тестування. Слід зазначити, що існує ймовірність прояву ризику порушення термінів і перевищення вартості тестування, внаслідок обмеженості проектних ресурсів, що негативно впливає не лише на процеси організації і управління тестуванням, але й на оптимальний обсяг тестування та впорядкування цілей тестування [1].

Одним із завдань сучасного вендора IT-компанії, що займається розробкою та продажем програмного забезпечення, є скорочення невиробничих витрат, у тому числі на зміст і обслуговування власної IT-інфраструктури. Ефективним способом вирішення цієї проблеми є перехід на застосування хмарних сервісів.

Хмарні обчислення – це новітній обчислювальний стандарт, який широко застосовується для підтримки завдань проектування програмного забезпечення, у тому числі, для його тестування.

Як показує практика, хмарне тестування, допомагає забезпечити потребу вендоров ПЗ в апаратних і програмних ресурсах за рахунок застосування гнучкої і відносно недорогої хмарної платформи для тестування. Проте важливо відмітити, що для реалізації широких можливостей хмарного тестування необхідно використати ефективну модель тестування.

Таким чином, розробка ефективної моделі хмарного тестування ПЗ є **актуальною задачею** та представляє науково-практичний інтерес.

Об'єктом дослідження магістерської роботи є процеси тестування програмного забезпечення.

Предметом дослідження є моделі хмарного тестування програмного забезпечення 1С.

Метою дослідження є підвищення ефективності тестування ПЗ, яке створене із використанням технологічної платформи 1С, шляхом розробки моделі хмарного тестування, що надає користувачеві можливість скоротити витрати в порівнянні із традиційними методами тестування.

Для досягнення поставленої мети в роботі необхідно виконати наступні завдання:

- проаналізувати сучасні технології хмарного тестування ПЗ та методологічні підходи до розробки моделей хмарного тестування ПЗ;
- розробити модель хмарного тестування ПЗ застосунків 1С та оцінити її ефективність;
- розробити програму для реалізації моделі хмарного тестування програмного забезпечення 1С.

Методи дослідження: об’єктно-орієнтованого аналізу та проектування ПЗ, хмарної обробки інформації та тестування

Наукова новизна одержаних результатів. Розроблено модель хмарного тестування ПЗ 1С, яка в порівнянні з традиційними методами тестування програмних продуктів забезпечує скорочення витрат на проектування та розробку ПЗ.

Практичне значення одержаних результатів. Розроблено програму для хмарного тестування застосунків 1С: Підприємство та 1С:Бітрикс

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ХМАРНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Основні положення і завдання хмарного тестування програмного забезпечення

Тестування програмного забезпечення – це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування – це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок [2]

IT-компанії, що проводять тестування в цілому й тестування на навантаження, тестування продуктивності та моніторинг виробництва, зокрема, стикаються з рядом таких проблем, як обмежений бюджет, дотримання термінів, високі витрати на тести, велику кількість тесткейсов, обмеження на повторне використання тестів і так далі [3].

Вирішити ці проблеми можна за допомогою хмарного тестування.

Хмарні обчислення стали новою обчислювальною парадигмою, в якій хмара може надавати видалено розміщені віртуальні апаратні і програмні ресурси і пропонувати користувачам модель обслуговування на вимогу.

Саме хмарне тестування стало новим підходом до тестування, завдяки якому середовища хмарних обчислень використовуються для застосування сучасного вигляду тестування, що забезпечує високу ефективність цього процесу при мінімізації витрат організацій-користувачів.

Хмарне тестування – це форма тестування програмного забезпечення, при якій тестування проводиться з використанням ресурсів через хмарні застосування в середовищі хмарного тестування. Ефективне необмежене сховище, швидка доступність інфраструктури з масштабованістю, гнучкість і доступність розподіленого середовища тестування скорочують час виконання тестування великих застосувань і призводять до економічно ефективних рішень.

Основними завданнями хмарного тестування є:

- підтвердження якості хмарних застосувань, включаючи їх функціональність, бізнес-процедури, продуктивністю і масштабованість, з урахуванням набору вимог, що пред'являються до додатків;
- перевірка хмарної сумісності в хмарній інфраструктурі.

Для запуску набору текст-кейсов в хмарному застосуванні необхідно виконати наступні дії:

- створення і налаштування хмарних обчислень;
- запуск хмарних обчислень;
- завантаження тестованих застосунків і тестових даних в хмару;
- запуск тестів користувача;
- отримання результатів тесту.

Такий процес займає багато часу і схильний до помилок, але його можна спростити шляхом виконання деяких тестів на хмарних машинах з автоматизацією.

Провідні хмарні провайдери і підприємства впроваджують центри обробки даних, спеціально призначені для хмарних обчислень.

Як показує практика, оператори центрів обробки даних, постачальники мережевого устаткування, виробники і постачальники високопродуктивних маршрутизаторів і комутаторів, облаштувань зберігання цих, обчислювальних платформ і засобів безпеки стикаються з новими проблемами в хмарному тестуванні при рішенні завдань оцінки і оптимізації.

Виділено мережу категорій хмарного тестування :

1 Віртуальна інфраструктура.

Віртуальна комутація дозволяє створювати такі топології мережі, як віртуальні машини, і зв'язувати їх на програмному рівні. Оскільки продуктивність сильно варіюється залежно від загального навантаження на сервер, дуже важливо протестувати масштабованість віртуального комутатора, щоб задовольнити величезні вимоги до пропускної спроможності на рівні комутації доступу.

Розраховані на багато користувачів хмари мають бути перевірені, щоб гарантувати, що угоди про рівні обслуговування можуть бути виконані при стресових подіях, таких як підвищена мобільність віртуальних машин.

2 Інфраструктура зберігання даних.

Трафік сховища використовує велику доступну смугу пропускання в конвергентній мережі LAN/ SAN і вимагає преференційного режиму. Ключові показники, включаючи пропускну спроможність введення-виводу, читання / запис, помилки і затримки, можуть використовуватися для вибору постачальників або налаштування сховища. Необхідно перевірити цільові локальні і мережеві сховища, конвергентні мережеві адаптери і комутатори.

3 Комутація рівнів.

Величезний об'єм трафіку проходить між додатками в центрі обробки даних на окремих фізичних серверах (зі сходу на захід). Оскільки тисячі користувачів одночасно дістають доступ до контенту / додаткам з Інтернету (з півночі на південь), вам необхідно перемикати рівні, які можна масштабувати. Тестування повинне перевірити масштабованість і відмовостійкість мережевих доменів.

4 Перехід на протокол IPv6.

Оскільки оператори центрів обробки даних стикаються з проблемою підтримки IPv6 в центрі обробки даних, необхідно виробити оптимальні рішення.

5 Безпека.

Додатки віртуальної безпеки - це віртуальні реалізації функцій безпеки, розподілені між компонентами хмарних застосувань. Вони служать для захисту

кожного компонента від іншого трафіку в загальних мережах і інших віртуальних машин на віртуалізованих серверах.

Щоб перевірити ефективність загальних заходів безпеки, використовується тестова система для відправки набору протоколів додатків між віртуальними машинами. При цьому слід переконатися, що трафік був заблокований або дозволений на основі налагоджених політик безпеки.

Облаштування захисту периметра, такі як брандмауери, системи запобігання вторгненням, уніфіковані системи управління загрозами і шлюзи VPN, можуть бути протестовані з використанням поєднання реального трафіку додатків і шкідливих програм для визначення ефективності, точності і продуктивності безпеки. Пікові місткості і швидкості з'єднання / транзакції / тунеля (IPsec) можна оцінити для вибору постачальника і налаштування мережі.

6 Наскрізна якість обслуговування (End - to - end QoS).

Будь-яка публічна або приватна хмара вимагає дотримання якості обслуговування від порту входу глобальної мережі на усьому шляху до окремих віртуальних машин. Багаторівнева архітектура забезпечує якість обслуговування для клієнтів і типів даних. Тестування і оцінка є важливими компонентами оптимізації усього центру обробки даних для конкретних застосувань і клієнтів, забезпечуючи найкращу загальну якість обслуговування.

7 Доставка додатків.

Постійно зростаюче використання додатків, насичених мультимедійними даними, призводить до того, що все більше число користувачів частіше стикаються з низькою якістю обслуговування.

Хмарний дата-центр має найкращу можливість вирішити дану проблеми завдяки еластичності смуги пропускання.

Щоб забезпечити якісну доставку додатків, необхідно протестувати архітектуру обчислювальної платформи і додатків, які можуть охоплювати безліч

фізичних і віртуалізованих серверів, використовуючи величезний об'єм трафіку між центрами обробки даних, а також величезний об'єм зустрічних відео-сесій.

1.2 Огляд основних видів хмарного тестування

Архітектура хмарних обчислень, як і будь-яке інше застосування або програмне забезпечення, складається з двох основних розділів:

- Front – End – це клієнт або будь-яке застосування, що використовує хмарні сервіси;
- Back – End – це мережа клієнтських машин з серверами, що мають комп'ютерну програму і систему зберігання даних.

Хмара має централізований сервер для адміністрування системних клієнтів, запитів тощо. Після розробки призначених для користувача сценаріїв, розробки та виконання тесту постачальник хмарних послуг надає результати й аналітику корпоративним IT-спеціалістам через інформаційні панелі в реальному часі для повного аналізу роботи їх застосувань в періоди пікових навантажень.

На рисунку 1.1 представлені основні види хмарного тестування [4].

Рисунок 1.1 - Основні види хмарного тестування

Розглянемо особливості основних видів хмарного тестування.

1.2.1 Функціональне хмарне тестування програмного забезпечення

Функціональне тестування ПЗ перевіряє усі функції програмного забезпечення і його взаємодію з устаткуванням. Для проведення функціональних випробувань тестувальники можуть використати такі інструменти, як Rapise, Sauce Labs і TimeShiftX. Ці хмарні інструменти тестування програмного забезпечення використовують наступні методи:

- тестування системи. Цей метод тестування оцінює відповідність системи функціональним і системним вимогам.
- приймальні випробування. Це доводить, що додаток задовольняє певні потреби користувачів.
- інтеграційне тестування. Це тестування гарантує, що додаток сумісний з різними платформами і добре працює при переході від однієї хмарної інфраструктури до іншої.

Функціональне веб-тестування може виконуватися як вручну тестувальником, так і автоматично з використанням програмного забезпечення.

Інструменти функціонально тестування програмного забезпечення, які використовуються для тестування звичайних застосувань, мають бути переоцінені стосовно додатка тестування, розміщеного в хмарі. Це важливо, оскільки потрібні інструменти, що дозволяють інженерам-тестувальникам аналізувати мережу, робочий стіл і наслідки змін в хмарі.

1.2.2 Нефункціональне хмарне тестування програмного забезпечення

Нефункціональне тестування також відоме як тестування продуктивності, оскільки воно дозволяє перевіряти нефункціональні аспекти програмного забезпечення, такі як його продуктивність, зручність використання і надійність. Для проведення цього типу тестування можна використати хмарні інструменти, такі як CloudTest, AppPerfect, CloudTestGo і AppLoader.

Ці інструменти тестування пропонують наступні типи нефункціонального тестування :

- тестування бізнес-вимог. Цей метод тестування перевіряє, наскільки точно додаток відповідає вказаним бізнес-вимогам. Цей метод також включає тестування хмарної доступності, яке гарантує відсутність часу простою додатка.
- тестування безпеки. Цей тип тестування потрібний для забезпечення безпечного зберігання і передачі даних. Механізми безпеки додатків тестуються відповідно до трьох критеріїв: ефективність, точність і продуктивність. Найбільш популярними інструментами для тестування безпеки в хмарі являються Nmap, Nessus і Wireshark.
- масштабованість і тестування продуктивності. Хоча хмарні рішення мають бути масштабованими на вимогу, цей тип тестування гарантує, що додаток працюватиме правильно з різною кількістю користувачів. Під час випробувань навантажень тестери вимірюють час відгуку програмного забезпечення, тоді як

система піддається збільшенню навантаження. Також необхідно перевірити, як додаток працюватиме при надмірному стресі, тому також необхідно виконати стрес-тестування.

Якщо необхідно виміряти затримку відповіді додатка після його розгортання в хмарі, можна провести тестування латентності (тестування на затримку).

1.2.3 Тестування працездатності

Тестування працездатності потрібне для перевірки того, чи дійсно користувачі дійсно отримують прикладні послуги на вимогу.

Щоб перевірити це, команда тестування може проводити наступні типи тестування:

- тестування на сумісність. Це тестування оцінює сумісність додатка з різними середовищами і платформами.
- аварійне відновлення. Тестування аварійного відновлення дозволяє оцінити час аварійного відновлення і переконатися, що додаток знову стає доступним користувачам з мінімальною втратою даних.
- багаторівневе тестування. Це тестування перевіряє, чи може додаток забезпечити достатній рівень безпеки і контролю доступу, коли декілька користувачів звертаються до нього в хмарі.

Вище перелічені тести хмарних застосувань можуть мати різні цілі і результати тестування.

Застосування того або іншого виду тестування визначається особливостями конкретного хмарного застосування і/або використовуваною моделлю.

1.3 Огляд і аналіз платформ хмарного тестування

Хмарні обчислення ускладнюють процес розробки зважаючи на величезну кількість компонентів, які повинні взаємодіяти між собою. Щоб гарантувати якість і

надійність, важливо мати чітке уявлення про платформи і послуги хмарних обчислень. В той же час еластичність і автоматизоване виділення ресурсів, які забезпечують хмарні технології, радикально покращують життєвий цикл розробки ПЗ [5].

Можливість виконання на вимогу тестів на масове навантаження або на проникнення дозволяє групам програмістів легко і дешево розробляти і тестувати масштабовані застосування.

Іншими словами, існує тісний взаємозв'язок між засобами розробки і розгортання з одного боку і можливостями для тестування з іншою.

Тому для розробки якісних моделей хмарного тестування необхідно враховувати особливості хмарної платформи, яка буде використана в якості середовища хмарного тестування.

Розглянемо характеристики популярних промислових платформ хмарного тестування.

1.3.1 Хмарна платформа Google Cloud

Google Cloud Platform (GCP), пропонована Google, є набір служб хмарних обчислень, які працюють на тій же інфраструктурі, яку Google використовує для своїх кінцевих користувачів, таких як Google Search і YouTube [6].

GCP написаний на мовах Java, З++, Python, Go, Ruby.

Разом з набором інструментів управління GCP надає ряд модульних хмарних сервісів, включаючи обчислення, зберігання даних, аналіз даних і машинне навчання.

Реєстрація вимагає кредитної карти або реквізити банківського рахунку.

Хмарна платформа Google надає сервіси IaaS, PaaS і безсерверные обчислювальні середовища.

GCP є частиною Google Cloud, яка включає інфраструктуру загальнодоступної хмари GCP, а також G Suite, корпоративні версії Android і Chrome OS, а також інтерфейси прикладного програмування (API) для машинного навчання і сервісів корпоративного мапинга.

Доступ до сервісів Google Cloud Platform можуть отримати розробники програмного забезпечення, адміністратори хмари і інші IT-спеціалисты підприємства через загальнодоступний Інтернет або через виділене мережеве з'єднання.

Google Cloud Platform пропонує сервіси для обчислень, зберігання, мереж, великих даних, машинного навчання і Інтернету речей (IoT), а також інструменти управління хмарою, безпеки і розробки (Рисунок 1.2).

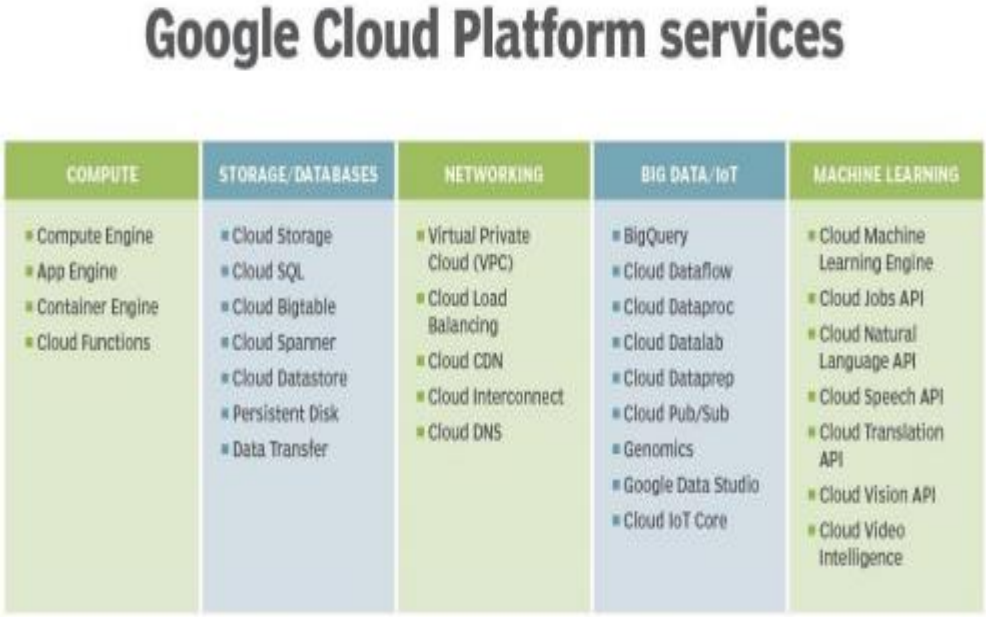


Рисунок 1.2 - Сервіси платформи Google Cloud

Основні продукти хмарних обчислень в Google Cloud Platform включають:

- Google Compute Engine, що є інфраструктурою як послугою (IaaS), яка надає користувачам екземпляри віртуальних машин для розміщення робочого навантаження;

- Google App Engine, що є платформою як послугою (PaaS), яка надає розробникам програмного забезпечення доступ до масштабованого хостингу Google. Розробники також можуть використати комплект розробника програмного забезпечення (SDK) для розробки програмних продуктів, працюючих на App Engine;

- Google Cloud Storage - платформа хмарного зберігання, призначена для зберігання великих неструктурованих наборів даних. Google також пропонує варіанти

зберігання бази даних, у тому числі Cloud Datastore для нереляційного сховища NoSQL, Cloud SQL для повністю реляційного сховища MySQL і нативну базу даних Cloud Bigtable від Google;

Google Container Engine - система управління і оркестровки контейнерів Docker, яка працює в загальнодоступній хмарі Google. Google Container Engine заснований на движку оркестровки контейнерів Google Kubernetes.

Google Cloud Platform пропонує послуги з розробки додатків і інтеграції.

Наприклад, Google Cloud Pub / Sub - це керована служба обміну повідомленнями в режимі реального часу, яка дозволяє обмінюватися повідомленнями між додатками.

Крім того, кінцеві точки Google Cloud дозволяють розробникам створювати сервіси на основі API RESTful, а потім робити ці сервіси доступними для клієнтів Apple iOS, Android і JavaScript. Інші пропозиції включають DNS -серверы Anycast, прямі мережеві з'єднання, балансування навантаження, послуги моніторингу і ведення журналів.

1.3.2 Хмарна платформа Amazon Web Services

Amazon Web Services (AWS) - хмарна платформа з широкими можливостями, яка надає 165 повнофункціональних сервісів для центрів обробки даних по усій планеті.

Мільйони клієнтів, у тому числі стартапы, що стали лідерами за швидкістю росту, найбільші корпорації і передові урядові установи, довіряють AWS в питаннях розміщення інфраструктури, підвищення гнучкості і зниження витрат (рис. 1.3)[7]

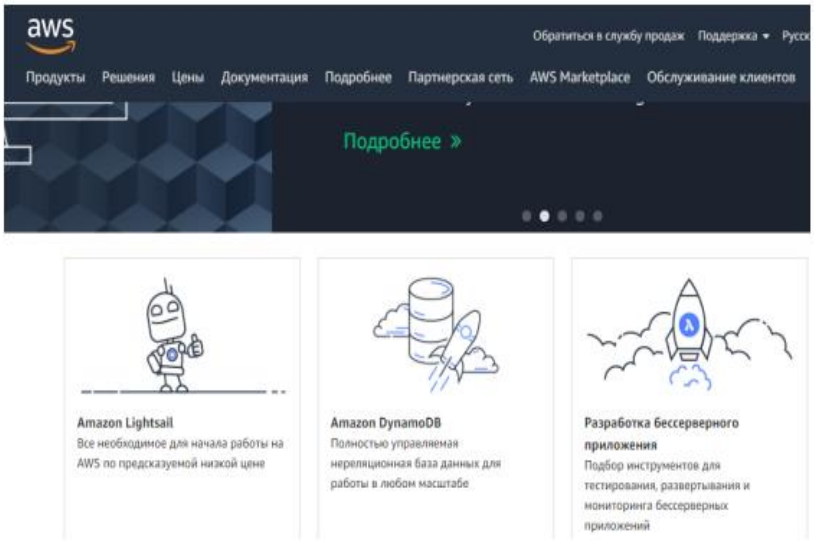


Рисунок 1.3 - Сервіси платформи Amazon Web Services

Переваги платформи :

1. Широкі функціональні можливості: AWS пропонує набагато більше сервісів і можливостей в їх рамках, чим будь-який інший постачальник хмарних рішень. Хмарна платформа AWS пропонує більше 165 повнофункціональних сервісів, у тому числі більше 40 унікальних.

2. Широка мережа клієнтів : на платформі AWS створена найбільша і найдинамічніша екосистема сервісів з мільйонами активних клієнтів і десятками тисяч партнерів по всьому світу. Клієнти з різним масштабом діяльності, у тому числі стартапы, корпорації і організації державного сектора, практично у будь-якій галузі використовують AWS в найрізноманітніших цілях.

3. Безпека: AWS - гнучке і захищене середовище для хмарних обчислень. Базова інфраструктура спроектована так, щоб задовольнити вимоги до безпеки міжнародних банків, установ у сфері оборони і інших організацій з високими вимогами до безпеки.

4. Надійність.

AWS - це ретельно продумані рішення, надійність, безпека і продуктивність.

Більше 12 років AWS поставляє хмарні сервіси мільйонам клієнтів по всьому світу для використання в різноманітних цілях, також пропонує самі кращі на ринку рішення для роботи у будь-якому масштабі.

В сукупності ці веб-сервіси хмарних обчислень надають набір примітивної абстрактної технічної інфраструктури і будівельних блоків, і інструментів розподілених обчислень.

Одним з таких сервісів є Amazon Elastic Compute Cloud, який дозволяє користувачам мати у своєму розпорядженні віртуальний кластер комп'ютерів, доступний постійно через інтернет. Версія віртуальних комп'ютерів AWS емулює більшість атрибутів реального комп'ютера, включаючи апаратні центральні процесори (CPU) і графічні процесори (GPU) для обробки, локальну / оперативну пам'ять, жорсткий диск / накопичувач; вибір операційних систем; мереж; і заздалегідь завантажене прикладне програмне забезпечення, таке як веб-сервери, бази даних, управління взаємовідносинами з клієнтами (CRM) і т. д.

1.3.3 Хмарна платформа Microsoft Azure

Microsoft Azure - це служба хмарних обчислень, створена Microsoft для створення, тестування, розгортання і управління додатками і службами через центри обробки даних, керовані Microsoft (рис. 1.4).

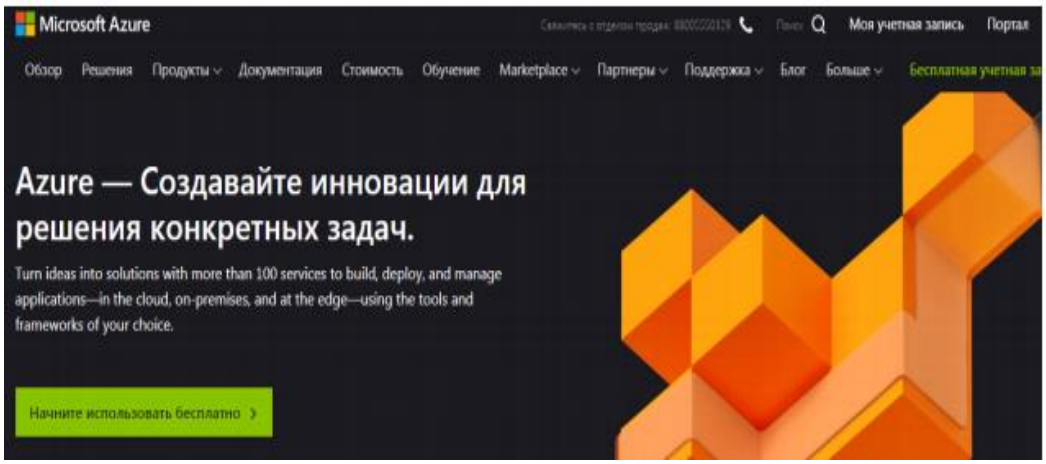


Рисунок 1.4 - Екран головної сторінки Microsoft Azure

Azure надає сервіси SaaS, PaaS і IaaS і підтримує безліч різних мов програмування, інструментів і середовищ, включаючи як програмне забезпечення і системи Microsoft, так і сторонніх виробників [8].

Azure використовує спеціалізован у операційну систему, звану MS Azure, для запуску свого «фабричного шару». Кластер, розміщений в центрах обробки даних Microsoft, який управляє обчислювальними ресурсами і ресурсами зберігання комп'ютерів і надає ресурси (чи їх підмножина) додаткам, працюючим поверх Azure.

Azure описана як «хмарний шар» поверх ряду систем Windows Server, які використовують Windows Server 2008 і налагоджену версію Hyper, - V, відому як гіпервізор Microsoft Azure, для забезпечення віртуалізації служб.

Масштабування і надійність контролюються Microsoft Azure Fabric Controller, який забезпечує відмову служб і середовища при збої одного або декількох серверів в центрі обробки даних Microsoft, а також забезпечує управління веб-додатком користувача, таким як виділення пам'яті. і балансування навантаження.

Azure надає API, заснований на REST, HTTP і XML, який дозволяє розробникові взаємодіяти із службами, Microsoft Azure, що надаються. Microsoft також надає клієнтську бібліотеку керованих класів, яка включає функції взаємодії із службами. Він також інтегрується з Microsoft Visual Studio, Git і Eclipse.

На додаток до взаємодії із службами через API користувачі можуть управляти службами Azure за допомогою веб-порталу Azure, який досяг загальної доступності в грудні 2015 року.

Портал дозволяє користувачам переглядати активні ресурси, змінювати налаштування, запускати нові ресурси і переглядати базовий моніторинг. дані з активних віртуальних машин і сервісів.

Azure - це набір служб хмарних обчислень, що постійно розширюється, який допомагає вашій організації вирішувати бізнес-завдання.

На думку фахівців Microsoft, платформа Azure надає розробникам свободу

створення, тестування і розгортання додатків, а також управління ними у великій глобальній мережі з використанням різних інструментів і платформ.

Вибір конкретної платформи для застосування в якості середовища хмарного тестування залежить від багатьох критеріїв: моделі хмарних обчислень, аналітики, зберігання, мережі, ціноутворення [9]. Так, для розробників, що використовують продукти корпорації Microsoft прийнятнішою видається платформа MS Azure.

1.4 Аналіз існуючих моделей хмарного тестування програмного забезпечення

1.4.1 Піраміда автоматизованого тестування

Автоматизоване тестування запропоноване М. Коном, який представляв процес автоматизації системи і перевірки ПЗ у формі піраміди.

Структура піраміди тестування зображена на рисунку 1.5 [10].

Рисунок 1.5 - Піраміда автоматизованого тестування

Піраміда автоматизованого тестування складається з наступних основних шарів (знизу-вгору) :

1 автоматизоване тестування модулів;

2 автоматизоване інтеграційне тестування;

3 автоматизоване наскрізне (end - to - end) тестування.

Як випливає з малюнка, наскрізне тестування знаходиться на самій вершині піраміди тестування, а модульне тестування утворює її основу.

Верхній компонент піраміди також включає тести призначеного для користувача інтерфейсу.

Елементи цієї піраміди відрізняються один від одного розміром, оскільки вони містять різну кількість тестових випадків, необхідних для виконання того або іншого типу тестування.

Крім того, на вершині піраміди є хмара. Саме ручне тестування представляється у вигляді хмари, оскільки вона не є невід'ємною і обов'язковою частиною піраміди.

Представлена піраміда типова для тестування програмного забезпечення. Але є багато його модифікацій. Усе залежить від типу тестованого застосування. Складові елементи піраміди, а також їх розташування можуть бути змінені.Іноді піраміда може бути змінена повністю.

1.4.2 Хмарний тест SOASTA

Хмарний тест SOASTA розгортається як сервіс на вимогу, використовуючи хмару для створення навантаження веб-сайтів [11].

Він включає послуги, що надаються тестувальниками навантаження, і платформи Global Cloud Test Platform, яка забезпечує кросс-облачную інфраструктуру для генерування навантаження.

Усередині додатка бібліотеки з відкритим початковим кодом є фундаментальною частиною пропозиції, використовуваної в усьому продукті для забезпечення різних функцій. SOASTA надає програмне забезпечення як частину сервісу [12]. Архітектура платформи SOASTA наведена на рис. 1.6.

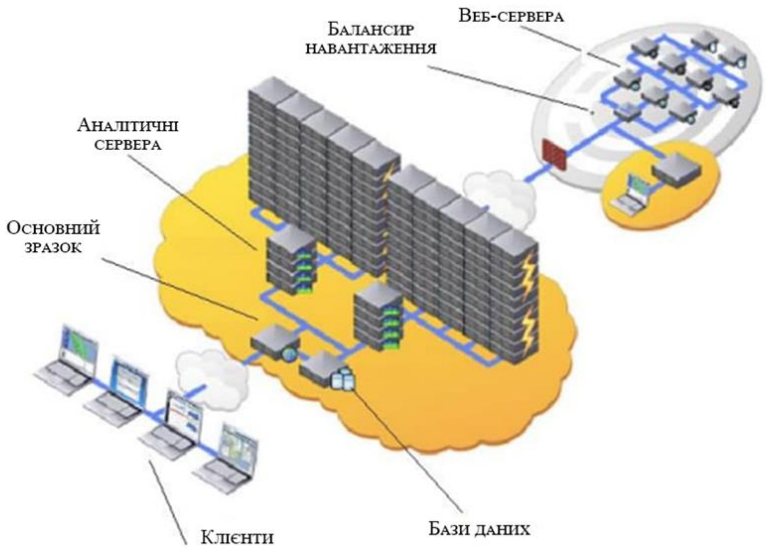


Рисунок 1.6 - Архітектура платформи SOASTA

Тоді як клієнти можуть використати додаток SOASTA для створення і виконання тестів, платформа Global Cloud Test Platform створена для підтримки додаткових інструментів, включаючи Apache JMeter, найпопулярніший інструмент для тестування навантаження з відкритим початковим кодом.

Платформа SOASTA зменшує складність і час розгортання сценаріїв JMeter в хмарі, що значно полегшує співтовариству JMeter створення, розгортання, виконання і аналіз тестів навантаження і продуктивності в масштабах мережі.

Скрипти JMeter працюють без змін. Після формування тесту SOASTA необхідно забезпечити управління і підготовку серверів і проведенні тесту.

Технологія забезпечення в реалізації SOASTA є однією з ключових особливостей платформи. У міру появи нових API, у тому числі альтернатив з відкритим початковим кодом, таких як libCloud, SOASTA використовуватиме їх для розширення охоплення Global Test Cloud.

Послідовність тестування веб-сайтів і застосунків за моделлю SOASTA представлена на рисунку 1.7

Рисунок 1.7 - Послідовність тестування веб-сайтів і застосунків за моделлю SOASTA

Інша ключова можливість - це аналітичний механізм в реальному часі, створений виключно для тестування веб-додатків і мобільних застосувань, що дозволяє командам по забезпеченню якості і розробці тестувати і контролювати свої веб-сайти як в типових, так і в екстремальних умовах трафіку. Враховуючи величезні об'єми даних, генерованих у веб-тестах, у тому числі моніторинг ресурсу під час виконання тесту, хмарний високо масштабований движок потрібний для надання корисної інформації в режимі реального часу.

Прикладом тесту, побудованого за представленою моделлю, є SOASTA TouchTest, що забезпечує повну автоматизацію функціонального тестування мобільних застосувань для сенсорних телефонів [12].

1.4.3 Модель тестування навантаження Visual Studio

Як приклад реалізації моделі хмарного тестування розглянемо сервіс Visual Studio/Team Foundation Services (TFS) на платформі Windows Azure.

TFS можна використати в якості бэк-энда для численних інтегрованих середовищ розробки (IDE), але він призначений для Microsoft Visual Studio і Eclipse на усіх платформах.

На рисунку 1.8 представлена структурна схема моделі тестування навантаження в Visual Studio [13].

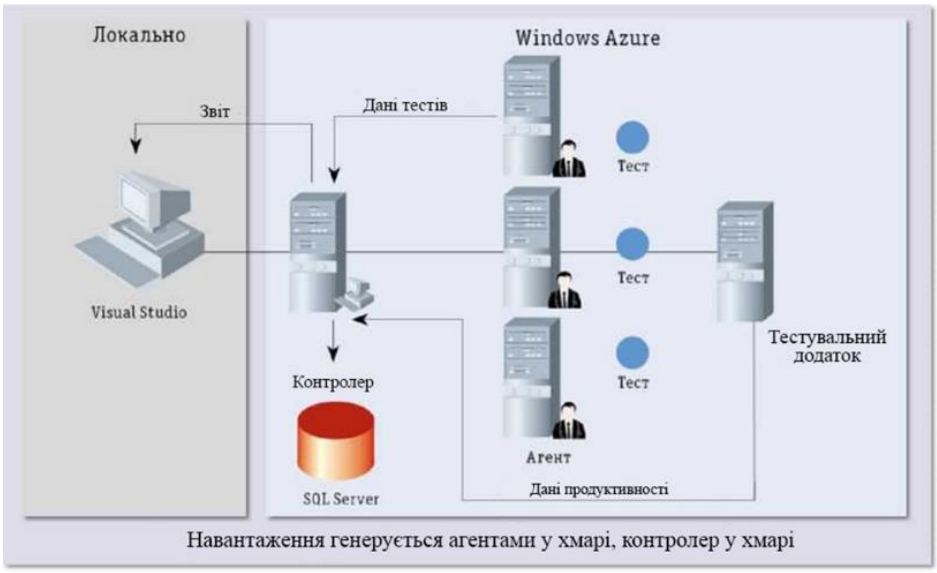


Рисунок 1.8 - Структурна схема моделі хмарного тестування навантаження в Visual Studio/TFS

Автоматизація процесу тестування виконується за допомогою компоненти Lab Management.

1.4.4. Модель Testing as a Service (TaaS)

Тестування як послуга (TaaS) - це модель аутсорсинга, в якій дії з тестування, пов'язані з деякими з бізнес-операцій організації, виконуються постачальником послуг, а не співробітниками [14]. TaaS може притягати консультантів для допомоги і консультування співробітників або просто передавати стороннім постачальникам послуг область тестування. TaaS найбільш підходить для спеціалізованих випробувань, які не вимагають глибоких знань дизайну або системи.

Послуги, які добре підходять для моделі TaaS, включають автоматичне регресійне тестування, тестування продуктивності, тестування безпеки, тестування основного програмного забезпечення ERP (планування ресурсів підприємства) і моніторинг / тестування хмарних застосувань.

TaaS також іноді називають «тестуванням на вимогу». Структурна схема моделі TaaS представлена на рисунку 1.9 [15].

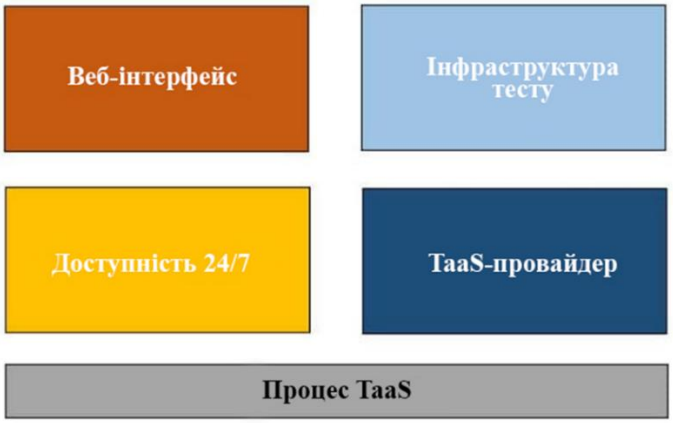


Рисунок 1.9 - Структурна схема моделі TaaS

Послідовність хмарного тестування ПЗ на основі моделі TaaS представлена на рисунку 1.10.

Рисунок 1.10 – Послідовність хмарного тестування ПЗ на основі моделі TaaS

TaaS рекомендується використати і проводити тестування при виникненні подібних проблем, вказаних нижче :

- тестування додатка має дуже короткий цикл виконання;
- тестування додатків вимагає великої автоматизації;
- завдання тестування не вимагає глибоких знань дизайну системи;
- команда тестування повинна провести спеціальне тестування;
- тестування додатків вимагає великих ресурсів для досягнення продуктивності або функціонального тестування.

Види TaaS :

- функціональне тестування як послуга. Це поведінкове тестування,

підтримуване TaaS Functional Testing. Він включає GUI -тестирование, інтеграційне тестування (SIT), регресійне тестування і Acceptance Testing або UAT -тестирование;

- тестування продуктивності як послуга. Тестування навантаження і стрес-тестування є частиною тестування продуктивності, яке може проводитися за допомогою тесту продуктивності TaaS. TaaS дозволяє створювати середовище реальних користувачів, які фактично є віртуальними користувачами, які вносять вклад у виконання навантаження і стресс-тестирования для тестованого застосування;

- тестування безпеки як послуга. TaaS також може виконувати тестування безпеки, просто виконуючи сканування уязвимостей в тестованих програмних застосуваннях і веб-сайтах.

Модель TaaS є кращим вибором для організацій, які хочуть понизити витрати на тестування.

1.4.5 Порівняльний аналіз моделей хмарного тестування

Результати порівняльного аналізу моделей хмарного тестування представлені в таблиці 1.1.

Аналіз відомих моделей хмарного тестування показав, що головним їх недоліком є недостатня універсальність і як наслідок, низька низька ефективність для деяких видів ПЗ.

Піраміда тестування є методикою хмарного автоматизованого тестування.

Так, технологія SOASTA призначена для тестування вэб-сайтов.

Сервіс TFS рекомендується передусім для Microsoft Azure.

Модель TaaS призначена для аутсорсингового тестування.

Крім того, в описах моделей немає відомостей про особливості застосування в умовах конкретної моделі хмарних обчислень.

Таблиця 1.1 - Порівняльний аналіз моделей хмарного тестування

Модель	Переваги	Недоліки
Піраміда автоматизованого тестування	- оперативність; - заощадження часу; - повторне використання; - Відстуність людського фактора; - автоматична звітність	- великі витрати на за соби автоматизації; - недостатня гнучкість; - можливі помилки тестування
SOASTA TouchTest	- можливість швидкого проектування, виконання, редагування і аналізу тест кейсів; - можливість тестування мультитач-жестов в нативних, гібридних і веб-додатках на iOS і Android; - безперервне тестування продуктивності мобільного застосування - TouchTest краще всього підходить для автоматизації критичних тест кейсів, які багаторазово використовуватимуться, оскільки кожен девайс має бути протестований окремо.	- мала точність масштабування; - мала точність синхронізації; - немає можливості управління устаткуванням. - відсутня опція симуляції вхідних дзвінків; - використовується тільки для веб- і мобільних застосувань.
Навантажувальне тестування MS Visual Studio	- генерування навантаження безпосередньо у хмарі; - підтримує усі типи робочих процесів тестування Visual Studio	- вимагає додаткових ресурсів для створення навантаження; - обмежена область застосування
TaaS	- гнучкість виконання тесту; - економія загальної вартості тестування; - швидкі результати тестування; - цілісність даних.	- організація повинна віддати на аутсорсинг свою інтелектуальну власність; - може знадобитися робота з групою підтримки хмари;

Таким чином, представляє актуальність розробка хмарної моделі тестування, більшою універсальністю, що володіє, ніж вищеописані.

1.5 Порівняльний аналіз сервісів хмарного тестування ПЗ

Для розуміння особливості моделей хмарного тестування, розглянемо сервіси тестування, засновані на моделях хмарних обчислень : SaaS, PaaS і IaaS.

Сервіс SaaS -тестування

Сервіс хмарних обчислень SaaS (програмне забезпечення як послуга) доступний клієнтам через інтернет і допомагає організаціям обходити складнощі, зв'язані з установкою ПЗ на комп'ютери [16].

Платформа для тестування SaaS - метод забезпечення якості ПЗ за рахунок всіляких перевірок, включаючи тестування продуктивності, безпеки, інтеграції даних, масштабованості, надійності і іншого.

Серед переваг методології тестування SaaS можна виділити наступні:

- велика надійність, масштабованість і доступність;
- скорочення витрат на установку і обслуговування ПЗ;
- швидке усунення помилок;
- швидке розміщення ПЗ;
- невисока плата за користування;
- мінімізація взаємозалежності системи на декількох рівнях;
- простота оновлення SaaS -системы.

До недоліків SaaS відносять:

- складність проведення тестування;
- програмні продукти розробляються набагато швидше, тому на перший план виходить забезпечення якості;
- для роботи з компонентами SaaS -приложений потрібно відповідна

кваліфікація IT-персоналу;

- середовище тестування повинне підтримувати автоматичні розміщення і валидацию застосування.

При SaaS -тестировании використовуються методи, які забезпечують надійну роботу додатка, створеного для цієї моделі.

Додатки, інфраструктура імережа вважаються ключовими компонентами SaaS -тестирования.

Методологія SaaS -тестирования застосовується для вирішення наступних завдань:

- тестування по стратегії білого і чорного ящика як частину модульного тестування;
- функціональне тестування для ретельної перевірки відповідності додатка вимогам;
- інтеграційне тестування проводиться для перевірки інтеграції SaaS - системы з іншими;
- дослідницьке тестування для нових тест-кейсов;
- тестування безпеки мережі, різних загроз, інтеграції і доступності;
- забезпечення якості з'єднання SaaS, а також тестування призначеного для користувача інтерфейсу, з акцентом на портативність і компактність;
- регресійне тестування для будь-якого оновлення, релиза або перенесення даних в додатку;
- тестування надійності робиться з метою понизити вірогідність помилок;
- перевірка безпеки мережі;
- тестування працездатності і масштабованості для перевірки, як поводиться додаток при піковому навантаженні в різних умовах
- перевірка сумісності додатка, з доступом на різних браузерах;
- тестування оновлень при додаванні нових функціональних особливостей або модернізації старих;

- тестування API для перевірки функціональності і безпеки;
- клієнтські запити, платежі і биллинг — складові етапи тестування в реальних умовах.

Складнощі, пов'язані з тестуванням SaaS -приложений :

- часті оновлення і релизы залишають мало часу на перевірку валидности і безпеці додатків;
- іноді компоненти бэкэнда, пов'язані з призначеним для користувача інтерфейсом застосування, залишаються без перевірки;
- забезпечення безпеки даних і запобігання витокам даних;
- важливо визначити найдоступніші зони і провести призначене для користувача тестування з участю як можна більшого числа людей з різних місць;
- під час інтеграції і перенесення SaaS -приложений нерідко виникають складності зі збереженням даних тестування;
- для нових релизов перевіряються усі аспекти, що відносяться до ліцензування, у тому числі кількість користувачів і функціональність додатка;
- відсутність стандартизації додатка.

Засоби SaaS -тестування:

1) PractiTest.

Цей інструмент тестування розроблений для того, щоб забезпечити клієнтів комплексними тестовими рішеннями і надати важелі контролю для процесу розробки додатків і тестування.

Функціональні особливості:

- забезпечення комунікації на різних рівнях;
- засоби управління проектом, загальним процесом тестування;
- у будь-який час можна упізнати статус проекту.

2) qTest.

Цей хмарний інструмент для управління тестуванням, який спрощує комунікації.

Функціональні особливості:

- простота взаємодії з учасниками команди;
- передбачена функція створення заміток, деталізовані звіти про дефекти;
- доступна пробна версія;
- інструмент дозволяє планувати і складати графік проекту, документацію по тест-кейсам, звітам і результатам;
- є зручна приладова дошка, на якій відображається досягнутий прогрес, запити і корисні звіти.

3) QMetry.

Інструмент дозволяє зв'язати вимоги проекту з його тест-кейсами і дефектами.

Функціональні особливості:

- QMetry зручний в умовах вимог, що швидко міняються, дозволяє використати старі тест-кейсы;
- результати і статус тест-кейсов можуть бути зафіксовані під час виконання тест-кейсов;
- сторінка виконання доступна для редактуры тест-кейсов в режимі реального часу;
- управління дефектами за допомогою посилання;
- усі дефекти, зафіксовані під час минулих тестів, можна легко виявити, так один і той же дефект не буде записаний двічі.

Cisco Web Ex, Google Apps, Consur — відомі приклади SaaS — додатків, які доступні за допомогою мережі і не вимагають додаткової установки.

Сервіс PaaS -тестування

У сервісі хмарних обчислень PaaS (платформа як послуга) розробники, по суті, орендують усе, що їм треба для створення додатка, покладаючись на хмарного провайдера для засобів розробки, інфраструктури і операційних систем. Це одна з трьох сервісних моделей хмарних обчислень [20]. PaaS значно спрощує розробку веб-додатків; з точки зору розробника, усе управління бэкэндом відбувається за кулісами.

Доступ до PaaS можна отримати через будь-яке інтернет-з'єднання, що дозволяє створити ціле додаток у веб-браузері.

Оскільки середовище розробки не розміщується локально, розробники можуть працювати з додатком з будь-якої точки світу. Це дозволяє групам, які розподілені по географічних місцях, співпрацювати. Розробники мають менший контроль над середовищем розробки, хоча це вимагає набагато менших витрат. PaaS надає швидкі, прості та економічні способи розробки, тестування і розгортання додатків. За допомогою стороннього постачальника є можливість керування операційними системами, віртуалізацією, сховищем, мережею, серверами і самим PaaS [21]. PaaS надає платформу з інструментами для тестування, розробки і розміщення додатків в одному середовищі. Постачальники послуг PaaS пропонують зокрема хмарні рішення для додатків .Net і Java. Популярні платформи PaaS : AWS Elastic Beanstalk, Windows Azure, Google App Engine, Apache Stratos та ін.

Сервіс IaaS -тестування

Хмарні сервіси IaaS, «інфраструктура як послуга», є моделлю самообслуговування для доступу, моніторингу і управління інфраструктурами видалених центрів обробки даних, такими як обчислювальні (віртуальні або фізичні), служби зберігання, мережі і мережі (наприклад, брандмауери).

Замість того щоб придбавати апаратне забезпечення безпосередньо, користувачі можуть придбавати послуги IaaS на основі аутсорсинга.

В порівнянні з SaaS і PaaS користувачі IaaS відповідають за управління додатками, даними, середовищем виконання, проміжним ПЗ і операційними системами. Постачальники як і раніше управляють віртуалізацією, серверами, жорсткими дисками, сховищами і мережами.

Багато провайдерів IaaS тепер пропонують бази даних, черги повідомлень і інші послуги вище за рівень віртуалізації. Іншими словами, користувачі IaaS отримують інфраструктуру, поверх якої вони можуть встановлювати будь-яку необхідну платформу. Користувачі несуть відповідальність за їх оновлення у разі випуску нових

версій. Обчислювальні і мережеві можливості роблять IaaS ідеальним місцем для запуску і управління циклами тестування і розробки.

Популярні платформи IaaS : Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE) та ін.

Порівняльний аналіз сервісів хмарного тестування

Результати порівняльного аналізу сервісів хмарного тестування представлено в таблиці 1.2.

Таблиця 1.2 - Порівняльний аналіз сервісів хмарного тестування

Сервіс	Характеристики середовища	Сфера застосування
SaaS	- централізоване управління; - розміщено на видаленому сервері; - доступно в мережі Інтернет; - користувачі не несуть відповідальності за оновлення устаткування або ПЗ	- стартапи або невеликі компанії, яким треба швидко запустити електронну комерцію при мінімальних затратах; - короткострокові проекти, які вимагають швидкої, простої і доступної співпраці; - додатки, які використовуються нечасто; - додатки, яким потрібний як веб, так і мобільний доступ
PaaS	- заснована на технології віртуалізації; - надає різні послуги, що допомагають в розробці і тестуванні, розгортанні додатків; - забезпечує розрахований на багато користувачів режим через одне і те ж застосування для розробки і тестування; - інтегрує веб-сервіси і бази даних	- оптимізація виробничих процесів, коли декілька розробників працюють над одним проектом розробки; - створення замовлених додатків; - швидка розробка або розгортання додатка
IaaS	- ресурси доступні як послуга; - вартість залежить від споживання; - сервіси добре масштабуються; - розрахований на багато користувачів режим; - організація збергає повний контроль над IT-інфраструктурою; - динамічність і гнучкість.	- компанії, які хочуть зьерегти контроль над своєю IT-інфраструктурою при мінімальних витратах на ресурси; - компанії, що швидко розвиваються.

Таким чином, вибір методології тестування визначається особливостями моделі хмарних обчислень, використовуваної конкретним вендором ПЗ.

Як показує практика, найбільш затребуваними для вирішення завдань хмарного тестування є методології SaaS і IaaS.

2 ПОБУДОВА МОДЕЛІ ХМАРНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 1С

2.1 Принципи побудови універсальної моделі хмарного тестування програмного забезпечення

Універсальна модель хмарного тестування (рисунок 2.1) складається з наступних етапів [22].

Рисунок 2.1 – Універсальна модель хмарного тестування

1. Формулювання мети тестування.

Хмарне тестування має як переваги, так і недоліки, тому отримати з нього вигоду можна тільки тоді, коли у компании-вендора

ПЗ є чітке розуміння потреб власного бізнесу. Тестування в хмарі вимагає тіснішої співпраці між розробниками і тестувальниками для проведення усіх необхідних тестів

упродовж усього життєвого циклу розробки програмного забезпечення.

2. Розробка плану тестування.

План тестування є описом області і діяльності хмарного тестування.

Перш ніж перемістити свій проект в хмару, необхідно визначити, які тести треба виконати, скільки часу вони прийматимуть і які риски при цьому можливі. Ця стратегія тестування дозволяє краще оцінити бюджет компанії і уникнути непередбачених витрат. [23]

Методика тестування повинна розвиватися в усіх цих сценаріях і повинна враховувати віртуалізовану інфраструктуру, мережу, бізнес-логіку додатків, дані і взаємодію з кінцевим користувачем.

Тестування хмарних застосувань вимагає тестування бизнеспроцессов, механізмів виключень, імітації сценаріїв відмов і сценаріїв аварійного відновлення.

3. Планування інфраструктури.

При створенні стратегії тестування слід також подумати про вимоги до інфраструктури, необхідні для створення тестового середовища. Необхідно переконатися, що хмарні служби надають необхідні засоби автоматизації тестування, програмне забезпечення, апаратне забезпечення і пропускну спроможність. Також важливо визначити, як довго знадобиться тестове середовище в цій конфігурації, і чи знадобляться які-небудь зміни в ній.

4. Вибір платформи і методології хмарного тестування.

При виборі хмарної платформи необхідно враховувати особливості розробки конкретного ПЗ і область його використання. Вибір методології тестування визначається використовуваною моделлю хмарних обчислень.

5. Тестування додатка.

На цьому етапі визначаються засоби автоматизації тестування.

6. Моніторинг і аналіз результатів тестування.

Хоча тестування в хмарі дозволяє забезпечити постійну доступність послуг, краще контролювати результати тестування в режимі реального часу. Аналізуючи

результати в реальному часі, тестувальники можуть швидко реагувати на проблеми, пов'язані з ефективністю або продуктивністю хмарного тестування.

2.2 Структурна схема плану хмарного тестування ПЗ 1С

В рамках магістерської роботи було розроблено структурну схему плану хмарного тестування ПЗ 1С8 та 1С:Бітрікс (рис.2.2).

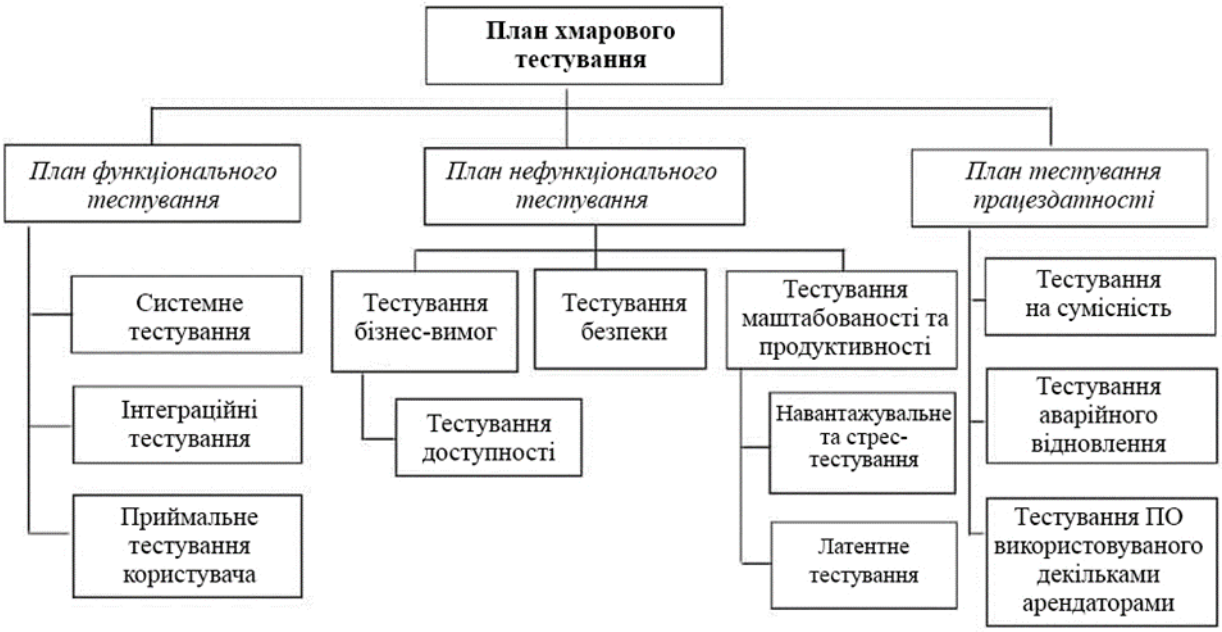


Рисунок 2.2 – Структурна схема плану хмарного тестування ПЗ 1С

Функціональне хмарне тестування виконується як для видалених, так і для локальних застосувань. Це тестування усіх функцій і властивостей системи, включаючи апаратне і програмне забезпечення. Воно проводиться на повній, інтегрованій програмній платформі, щоб перевірити її відповідність вимогам.

При функціональному хмарному тестуванні процес перевірки виконується відповідно до специфікацій системи або вимог в хмарі замість локального тестування ПЗ. Функціональне тестування не є досить повним, щоб визначити усі комбінації сайту і його працездатність в стресових умовах.

До функціонального тестування відносяться:

- системне тестування;
- інтеграційне тестування;
- призначене для користувача приймальне тестування.

Нефункціональне тестування проводиться, щоб переконатися в тому, що веб-додаток відповідає заданим вимогам до продуктивності. У хмарі область масштабованості додатків набагато ширша, ніж в традиційних моделях тестування продуктивності ПЗ.

До нефункціонального тестування відносяться:

- тестування бізнес-вимог;
- тестування безпеки;
- тестування масштабованості і продуктивності.

Хмарна масштабованість є ще однією серйозною проблемою, що вимагає тривалого тестування. Рішення хмарних обчислень завжди претендують на масштабованість на вимогу. Навантаження або стрес-тестування можна використати, щоб довести, що розроблене хмарне рішення можна масштабувати відповідно до вимог за допомогою програмних інструментів.

Отже, хмарне рішення може бути точно оцінене, а його потужність перевірена. Методики тестування продуктивності хмари дозволяють вимірювати продуктивність хмарних систем досить точно.

Тестування продуктивності з використанням методів тестування навантаження дозволяє отримати точне уявлення про можливості рішення в хмарі.

Продуктивність зазвичай пов'язана з можливостями додатка в хмарній інфраструктурі. Виявлення порогів, вузьких місць і обмежень є частиною тестування продуктивності. Для цього необхідно тестувати продуктивність при певному робочому навантаженні і варіювати характер трафіку на вимогу.

Цей вид тестування включає:

- хмарне тестування навантаження і стресове;

- латентне тестування.

Тестування навантаження додатка припускає створення великого призначеного для користувача трафіку і вимір його відгуку. Також необхідно настроїти продуктивність будь-якого застосування відповідно до певних стандартів.

Необхідно вимірювати час відгуку і виявляти проблеми, пов'язані з конкретними діями, тоді як система піддається зростаючому навантаженню з різних місць і розрахованим на багато користувачів операціям.

Слід обов'язково виявляти проблеми, оскільки система тестується до межі максимальної очікуваної місткості або часто за межами очікуваного використання.

Стрес-тест використовується для визначення здатності додатка підтримувати певний рівень ефективності за межами критичної точки або максимальної очікуваної місткості або понад очікуване використання.

Для будь-якого застосування важливо працювати навіть при надмірному стресі і підтримувати стабільність. Стрес-тестування гарантує це шляхом створення пікових навантажень з використанням тренажерів. Але вартість створення таких сценаріїв величезна.

Хмарне тестування використовується для виміру затримки між дією і відповідною відповіддю для будь-якого застосування після його розгортання в хмарі.

Тестування працездатності проводиться для того, щоб хмарне середовище могло надавати свої послуги користувачам за запитом. У цій категорії перевіряється сумісність, функціональна сумісність і розрахована на багато користувачів здатність середовища хмарних обчислень.

До цього виду тестуванню відносяться:

- тестування на сумісність;
- тестування аварійного відновлення;
- тестування ПЗ, використовуване декількома орендарями.

2.3 Модель хмарного тестування застосунків 1С8

На основі універсальної моделі хмарного тестування (рис. 2.1) була розроблена модель хмарного тестування застосунків 1С8, яка представлена на рис. 2.3

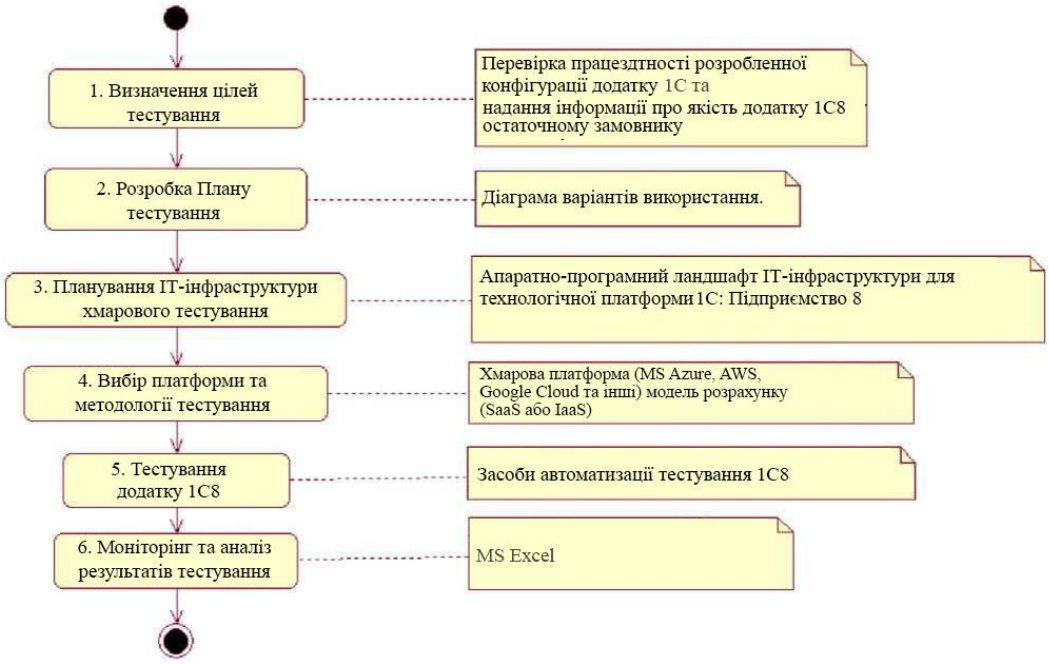


Рисунок 2.3 – Модель хмарного тестування застосунків 1С8

Опишемо кожен етап даної методики.

1. Мета тестування: перевірка працездатності розробленої конфігурації програми 1С8 і надання інформації про якість додатки 1С8 кінцевому замовнику.

2. Розробка плану тестування.

Тестування програми 1С8 включає:

- тестування конфігурації програми 1С8;
- тестування серверної частини програми 1С8.

Як методичну основу тестування конфігурації програми 1С8 будемо використовувати методику Apdex. Apdex – відкритий міжнародний стандарт, розроблений з метою формування об'єктивної оцінки показників продуктивності корпоративних інформаційних систем [24].

Така методика дозволяє:

- привести до простого значенням різнорідні фактори і безліч статистичних даних про продуктивність. Головна перевага методики в простому результаті, для швидкої оцінки стану продуктивності інформаційної системи;

- ранжувати операції по пріоритетності з точки зору бізнесу, що дозволяє правильно акцентувати увагу при моніторингу та оптимізації великої кількості операцій;

- побудувати індекс на підставі фактичних даних, отриманих при роботі всіх користувачів програми. Результуюча оцінка продуктивності за методикою Apdex є спільною, фактичної і об'єктивною.

Apdex є числовий мірою задоволеності користувачів продуктивністю додатків.

Для розрахунку Apdex збирається множина статистичних даних про час виконання операцій додатком.

Як методичної основи тестування серверної частини програми 1С8 використовується тест Гилева (ТРС-1С) [25].

Тест Гилева в умовних одиницях умовно показує можливу продуктивність системи 1С – чим вище значення, тим вище потенційна продуктивність.

В основу даного тесту покладено підхід до оцінки продуктивності сервера, заснований на тому, що остання визначається не завантаженістю і чергами до процесора, а здатністю виконати кількість операцій в одиницю часу.

3. Планування інфраструктури.

Апаратно-програмний комплекс IT-інфраструктури середовища тестування визначається відповідними вимогами технологічної платформи 1С: Підприємство 8.

4. Вибір платформи та постачальника хмарного сервісу.

Основні критерії вибору таких сервісів:

- надійність партнера, приналежність до брендових вендорів;

- можливість створення потрібної архітектури роботи;

- розміщення;

- продуктивність сервісів;

- вартість орендованих ресурсів.

Платформа 1С8 орієнтована на роботу в ОС Window s, томурекомендується віддати перевагу MS Azure. Як методології обираємо SaaS або IaaS.

5. Тестування додатка 1С8. Для автоматизації використовуються вбудовані засоби автоматизації тестування технологічної платформи 1С: Підприємство 8.

6. Моніторинг і аналіз результатів тестування.

Для збору, обробки і візуалізації результатів хмарного тестування рекомендується використовувати табличний процесор Excel.

За допомогою пропонованої моделі (рис.2.3) визначимо показники роботи з обліковою програмою 1С8 в хмарному середовищі MS Azure, що представляє IaaS - сервіс.

Для проведення тесту на хмарній платформі була створена віртуальна IT-інфраструктура, що має конфігурацію, представлену в таблиці 2.1 [26].

Таблиця 2.1 - Характеристики віртуальної IT-інфраструктури

Ресурс	Роль
Віртуальна машина F8 v2 (8 vCPU, 16 ГБ ОЗУ)	Сервер 1С + SQL Server
Сховище Premium SSD 512 ГБ	Диск для баз даних 1С
Віртуальна машина E4 v3 (4 vCPU, 32 ГБ ОЗУ)	Сервер RDS (служба видаленого столу)
Сховище Standard HDD 512 ГБ	Диск для профілів користувачів
Шлюз VPN	VPN Site - to - Sit

Конфігурація сервера була вибрана виходячи з калькуляції необхідних

ресурсів для роботи 10 користувачів з конфігураціями 1С : Управління Торгівлею 11.3.

На тестовий сервер був встановлений сервер 1С підприємства, СУБД SQL Server Standard 2017 і товстий клієнт 1С. Протокол взаємодії сервера 1С і SQL - сервера - Shared Memory, що за умовчанням використовується при розміщенні сервера 1С і SQL -сервера на одній платформі.

На тестовому сервері розміщена дві бази: база для тесту Гилева, Управління Торгівлею 11.3 – реальна база, в яких можуть працювати користувачі [27].

У конфігурацію «Управління Торгівлею» була вбудована система Apdex і сценарій тестування по виконанню стандартних операцій, які роблять користувачі щодня в подібних конфігураціях.

Для конфігурації 1С:Управление торгівлею 11.3 це:

- проведення повернення товарів від клієнта;
- проведення повернення товарів постачальникові;
- проведення замовлення клієнта;
- проведення перерахунок товарів;
- проведення вступ ТУ;
- проведення реалізації ТУ;
- проведення РКО.

Розроблений сценарій, по якому запускалося 10 віртуальних робочих місць, які одночасно виконували ряд операцій за сценарієм, підсумком якого був вимір час виконання кожної з операції.

Кожен тест на платформі проводився 3 рази в різний проміжок часу, щоб виключити чинник "навантаження" платформи, на якій тестувався додаток 1С8.

Результати тестування по тесту Гилева представлені в таблиці 2.2.

Таблиця 2.2 - Підсумки тесту Гилева

Перелік тестів (середнє значення за підсумками серії з 3 тестів)	MS Azure
Проведення тестів Г илева, в умовних одиницях	25,5

Цей результат відноситься до «зеленої» зони результатів тесту Гилева і вважається хорошим для цього типу завдань.

Слід врахувати, що відносний показник і, виходячи з практики, дуже чутливий до частоти процесора, який використовується на платформі, як власне і система 1С8.

Наступним етапом було проведено тестування за моделлю Apdex на конфігурації Управлінні Торгівлею 11.3.

Результати представлені в таблиці 2.3 і рисунку 2.4, відповідно.

Таблиця 2.3 - Підсумки тестів Apdex, 1С:Управление торгівлею 11.3

Перелік тестів	MS Azure
Проведення і повернення від клієнта, секунди	5,15
Проведення і повернення товарів постачальникові, сек	3,69
Проведення замовлення клієнта, сек	0,96
Проведення перерахунку товарів, сек	0,39
Проведення поступления ТУ , сек	3,90
Проведення реалізації ТУ, сек	3,35
Проведення РКО, сек	1,86

Слід зазначити, що система 1С є дуже складним програмним продуктом, конструктором, з елементів якого збирається інформаційна система підприємства.

Дуже важливо уміти правильно підібрати під цю інформаційну систему

апаратну платформу, яка б відповідала усім вимогам програмного забезпечення.

Саме тому необхідно використати спеціалізована платформи для досягнення максимальної продуктивності в даних тестах.

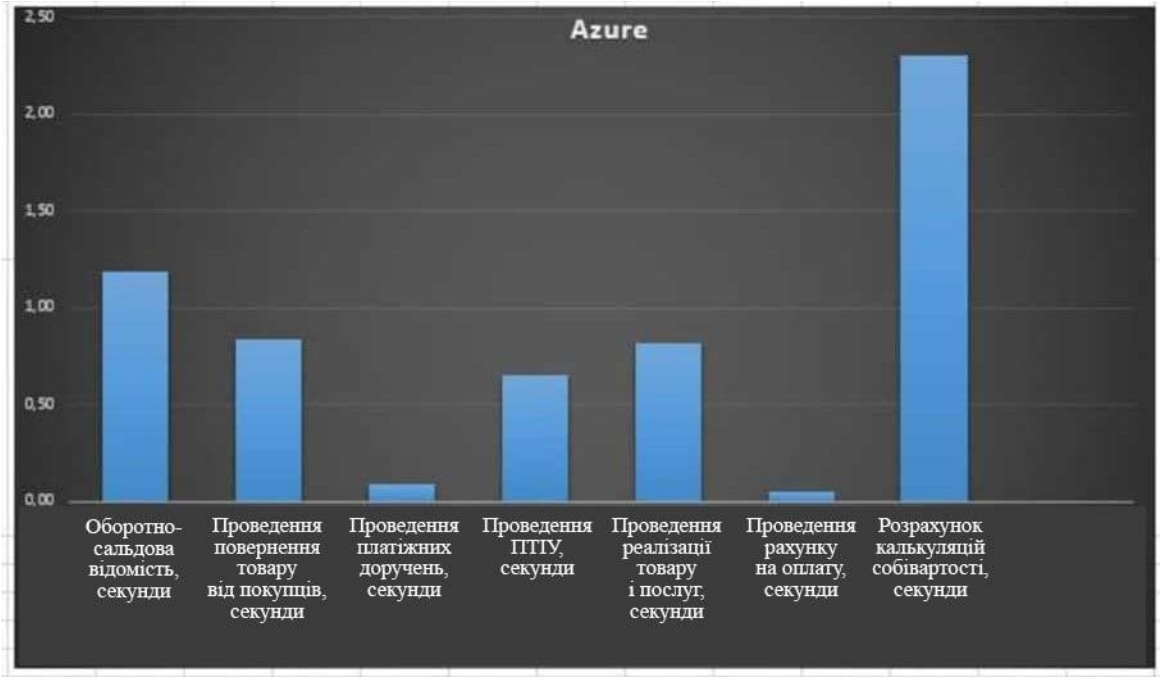


Рисунок 2.4 – Результати тестування за методикою Apdex конфігурації 1С: Управління Торгівлею

2.4 Модель хмарного тестування застосунків 1С:Бітрікс

На основі універсальної моделі хмарного тестування (рис. 2.1) була розроблена модель хмарного тестування застосунків 1С:Бітрікс, яка представлена на рис. 2.5

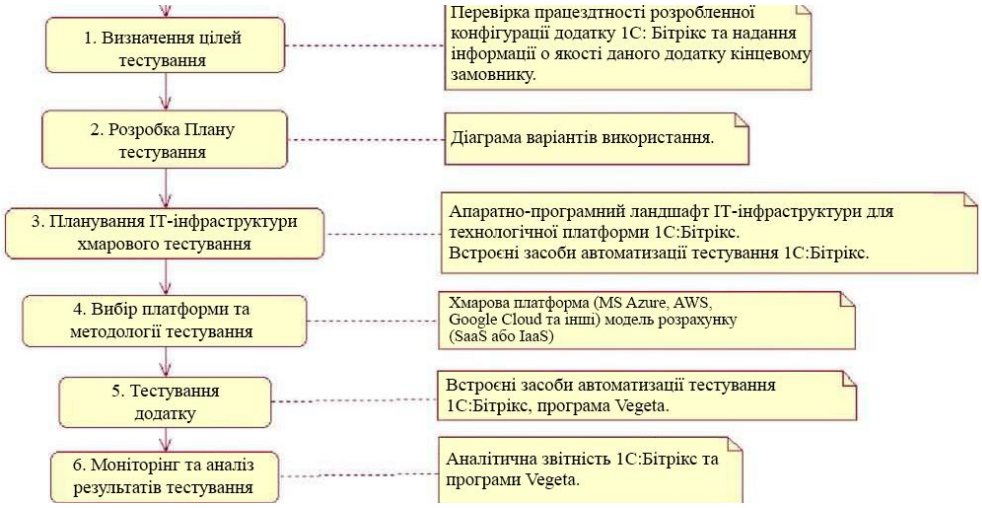


Рисунок 2.5 – Модель хмарного тестування застосунків 1С:Бітрікс

Опишемо кожен етап представленої моделі.

1. Мета тестування: перевірка працездатності розробленої конфігурації програми 1С: Бітрікс і надання інформації про якість даного продукту кінцевому замовнику.

2. Розробка плану тестування.

Тестування програми 1С: Бітрікс включає в себе наступні види тестування:

- тестування веб-додатки;

- тестування серверної частини.

Для автоматизації тестування серверної частини використовується програма-бенчмарк Vegeta [28].

Vegeta – це безкоштовна утиліта командного рядка для тестування HTTP-сервісів, написана на мові Go. Її можна підключити як бібліотеку для створення своїх інструментів навантажувального тестування.

3. Планування інфраструктури.

Апаратно-програмний комплекс IT-інфраструктури середовища тестування визначається відповідними вимогами платформи 1С: Бітрікс, програма Vegeta.

4. Вибір платформи і постачальника хмарного сервісу здійснюється відповідно до рекомендацій, наведених вище. Як методологій вибираємо SaaS або IaaS.

5. Тестування додатка 1С: Бітрікс. Для автоматизації використовуються вбудовані засоби автоматизації тестування 1С: Бітрікс і програма Vegeta.

6. Моніторинг і аналіз результатів тестування.

Для збору, обробки і візуалізації результатів хмарного тестування використовується вбудовані засоби 1С: Бітрікс і програми Vegeta, яка дозволяє створювати звіти в текстовому і графічному форматі.

Результати хмарного тестування IT-рішення на основі додатка 1С:Бітрікс.

В якості хмарної платформи вибрана платформа MS Azure, що надає сервіси IaaS і SaaS.

1 Тестування швидкості сайту (рисунок 2.6).

Ця сторінка відображає статистику по завантаження сайту, що збирається під час роботи користувачів з сайту.

Функції:

- відображає швидкість завантаження для користувачів з різних регіонів;
- дозволяє упізнати мінімальну, середню і найбільшу швидкість завантаження сторінок;
- дозволяє побачити, які підсистеми впливають на швидкість отрисовки сторінки : швидкість віддачі від сервера, dns -сервер, обробка html, інша обробка

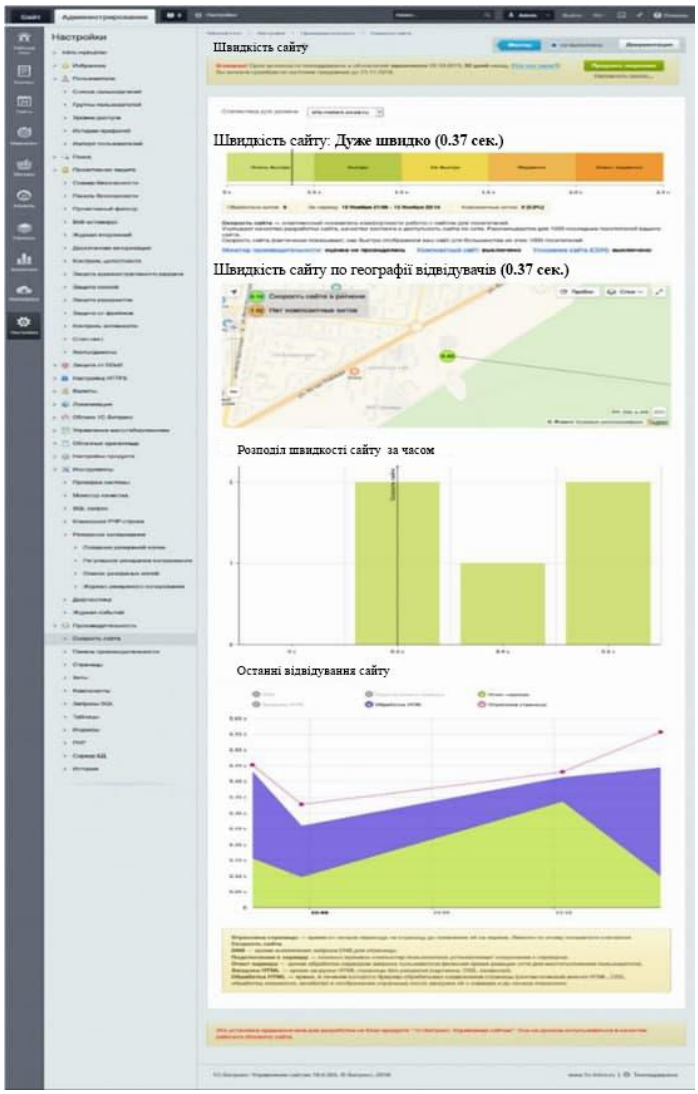


Рисунок 2.6 - Сторінка тестування швидкості сайта 1С:Бітрікс 2

2 Тестування конфігурації.

Дозволяє визначити, наскільки вистачає хостингу для оптимальної роботи системи. Перевіряються процесор, пам'ять, файлова підсистема. Також перевіряється робота php, пошти, MySQL (рисунок 2.7)

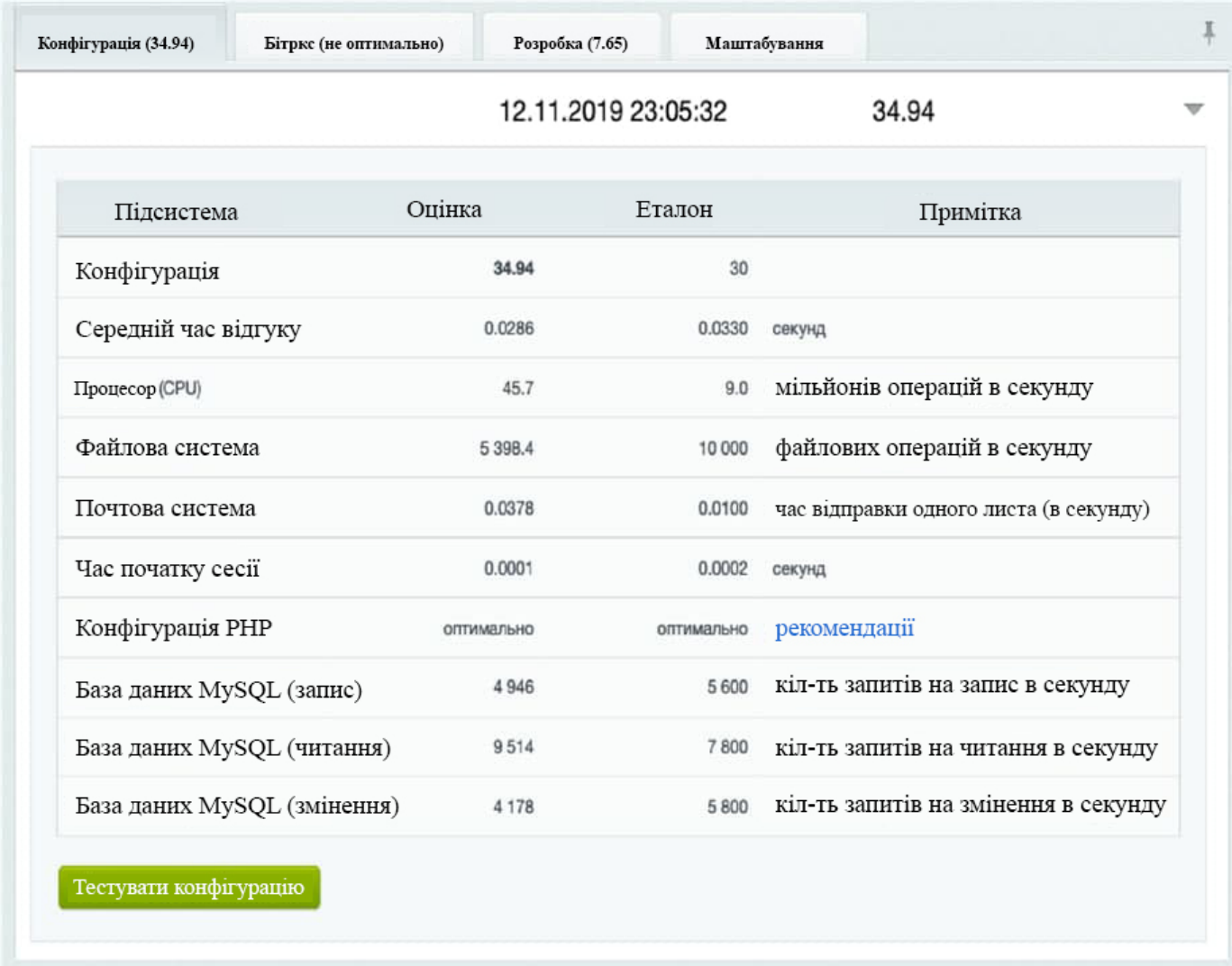


Рисунок 2.7 - Сторінка тестування конфігурації вебзастосунку

3 Тестування продуктивності сторінок сайту (рис. 2.8).

Запускається на панелі розробки. Дозволяє знайти повільні сторінки сайту, виявити, як елементи на сайті уповільнюють її роботу, знайти відключений кеш для «важких» сторінок та ін.

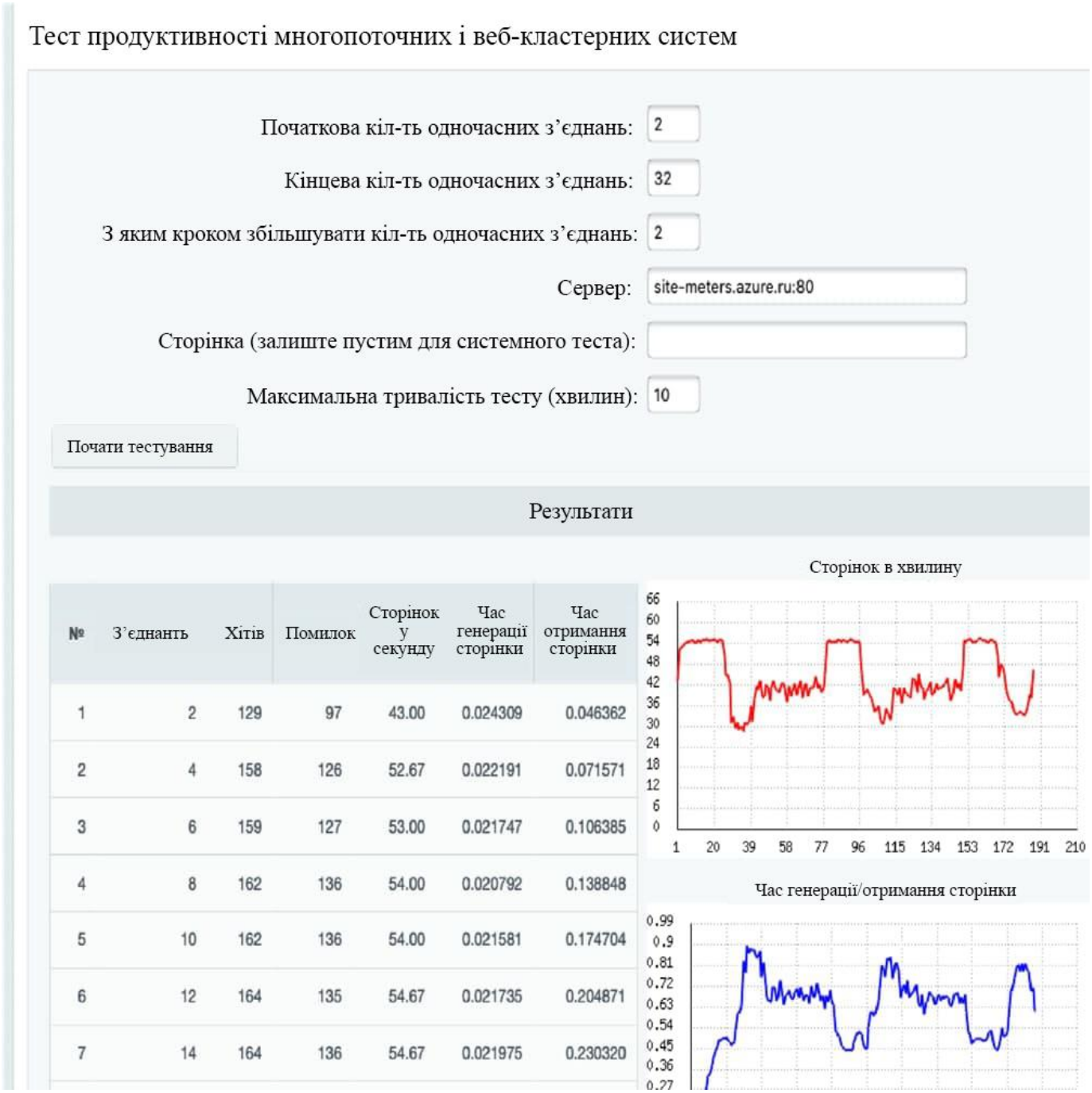


Рисунок 2.8 – Фрагмент сторінки тестування продуктивності сторінок сайту

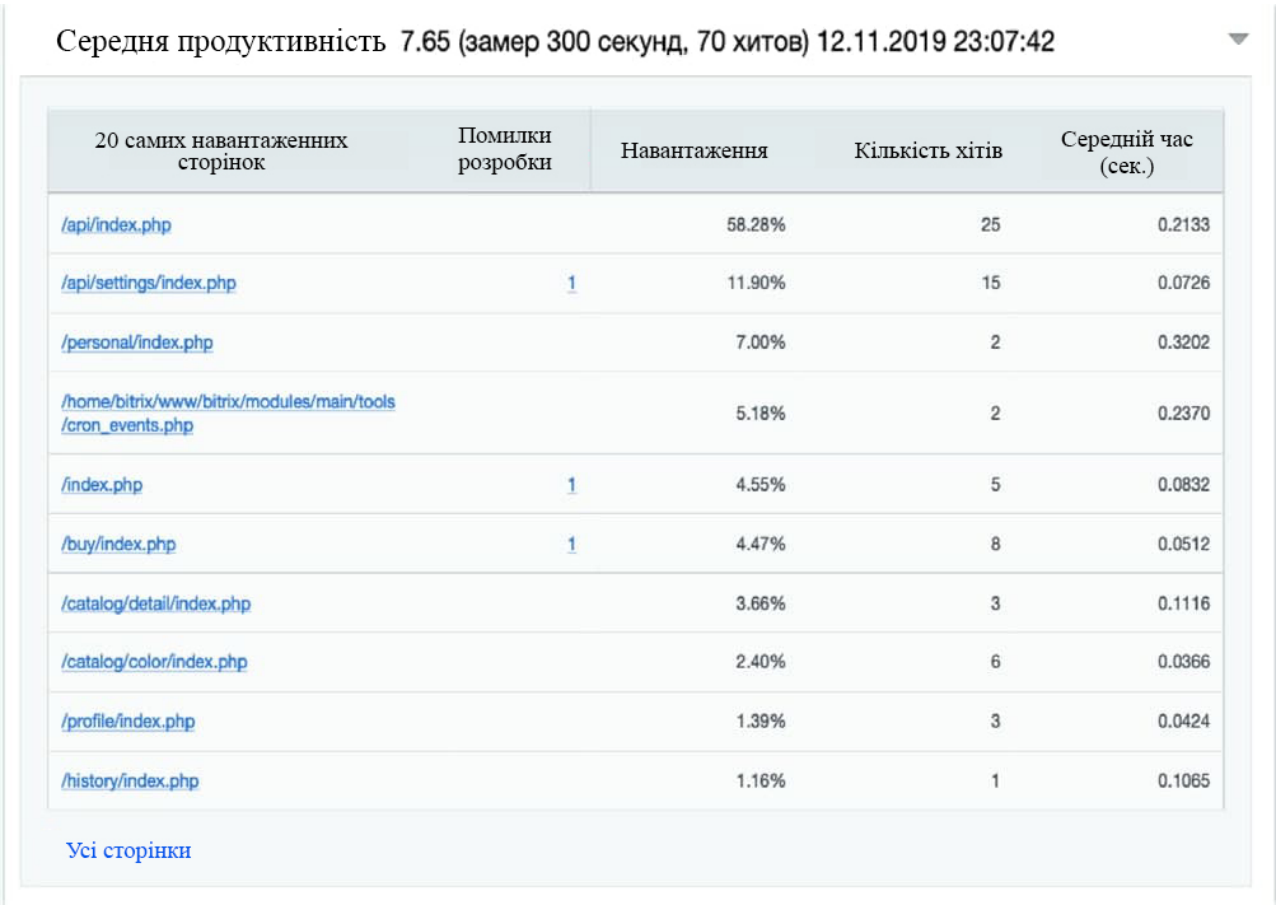
4 Тест на масштабованість.

Дозволяє отримати для одне або багато серверних рішень базові метрики по сторінок в секунду, що віддаються, і часу генерації сторінки (рисунок 2.8).

5 Тестування навантаження серверів.

Для тестування навантаження на сервер краще використати зовнішні застосування для тестування.

Оскільки вони краще дозволяють вичислити максимальні можливості сервісу, зайві ресурси сервера не витрачаються на систему моніторингу усередині сервісу.



Середня продуктивність 7.65 (замер 300 секунд, 70 хитов) 12.11.2019 23:07:42

20 самих навантаженних сторінок	Помилки розробки	Навантаження	Кількість хітів	Середній час (сек.)
/api/index.php		58.28%	25	0.2133
/api/settings/index.php	1	11.90%	15	0.0726
/personal/index.php		7.00%	2	0.3202
/home/bitrix/www/bitrix/modules/main/tools /cron_events.php		5.18%	2	0.2370
/index.php	1	4.55%	5	0.0832
/buy/index.php	1	4.47%	8	0.0512
/catalog/detail/index.php		3.66%	3	0.1116
/catalog/color/index.php		2.40%	6	0.0366
/profile/index.php		1.39%	3	0.0424
/history/index.php		1.16%	1	0.1065

Усі сторінки

Рисунок 2.9 – Фрагмент сторінки тестування на масштабованість

Для цій меті використовуємо додаток Vegeta.

Слід зазначити, що в продукті «1 С-Бітрікс: Управління сайтом» реалізований механізм, який дозволяє підключати до сайту будь-які «хмари» і легко управляти ними.

Розглянемо звіти результатів тестування з допомогою програми Vegeta.

Графіки результатів тестування навантаження для сторінок з включеним

проактивним захистом (дозволяє відсікати занадто часте звернення до сервера від користувача) приведені на рисунку 2.10

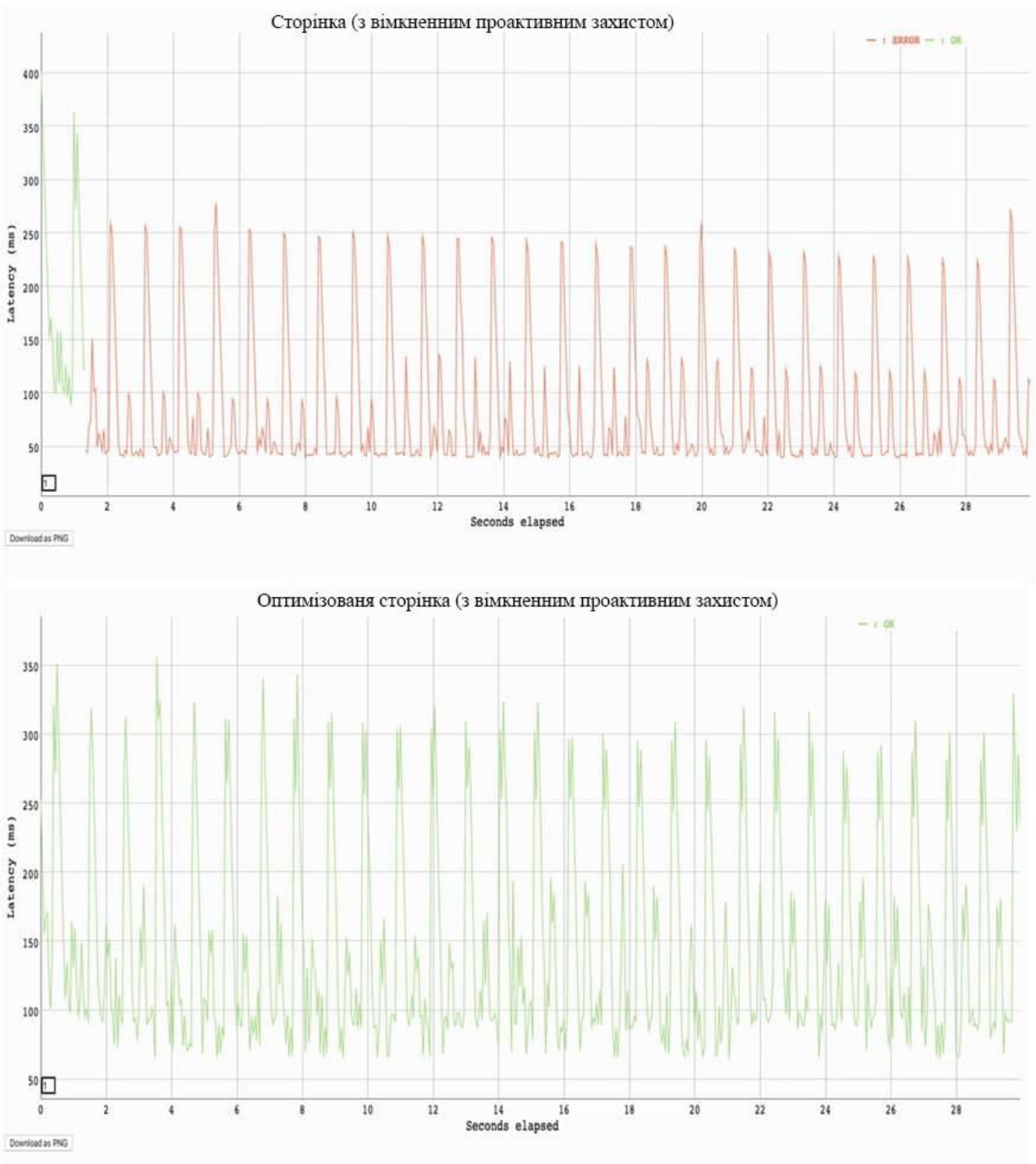


Рисунок 2.10 – Графіки результатів тестування навантаження для сторінок з включеним проактивним захистом

Графіки результатів тестування навантаження для сторінок з відключеним

проактивним захистом приведені на рисунку 2.11

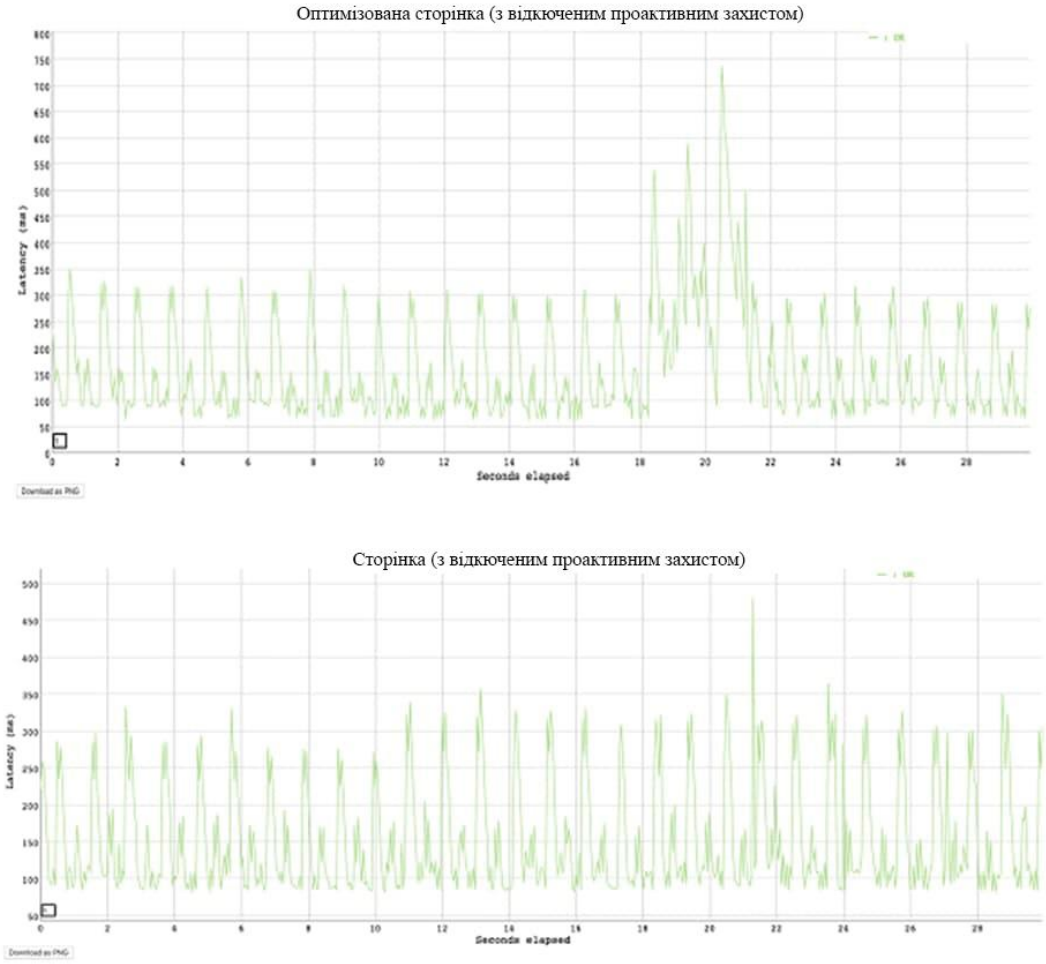


Рисунок 2.11 – Графіки результатів тестування навантаження для сторінок з відключеним проактивним захистом

Таким чином, пропонована модель забезпечує тестування різних видів ПЗ, що підтверджує її універсальність і ефективність.

3 ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ХМАРНОГО ТЕСТУВАННЯ ЗАСТОСУНКІВ 1С

При використанні об' єктно – орієнтованої методології для візуалізації компонентів, зв' язків та процесів розроблюваного програмного забезпечення використовуються засоби UML [29].

3.1 Ескізний проект програмного забезпечення

Для побудови складної програмної системи необхідним етапом є проектування.

Головною метою моделювання на етапі ескізного проектування є відображення процесу створення програмного забезпечення, визначення взаємозв'язків між модулями ПЗ у вигляді візуалізації в діаграмах.

Для проектування розроблюваного ПЗ було обрана мова моделювання UML систему Draw.io.

3.1.1 Розробка моделі варіантів використання

Для графічного відображення майбутнього ПЗ побудуємо діаграму варіантів використання. Обов'язковими елементами такої діаграми є: прецеденти, актори або дійові особини і безпосередньо відносини між акторами та варіантами використання.

Модель варіантів використання ПЗ представлена на малюнку 3.1 з Актором «Тестувальник».

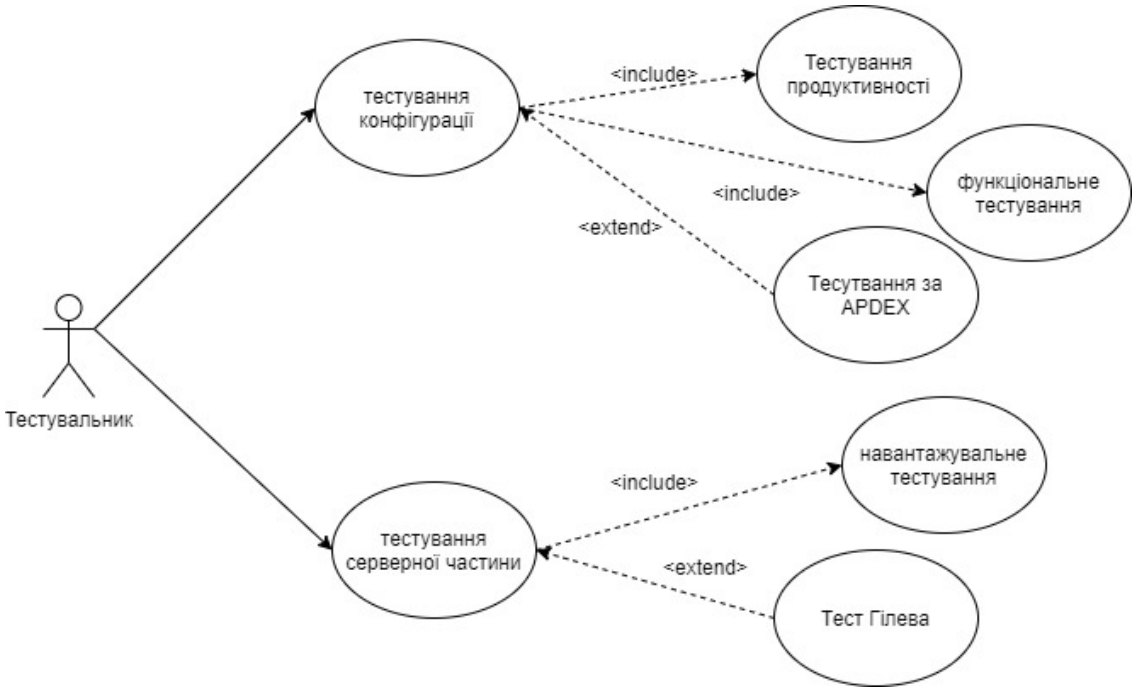


Рисунок 3.1 — Діаграма варіантів використання ПЗ

Функції користувача «Тестувальник» : виконує тестування конфігурації 1С (тест продуктивності, функціональне тестування); тестування серверної частини ПЗ (навантажувальне тестування, тестування Гілева).

3.1.2 Розробка специфікації варіантів використання

Потік подій варіанта використання має такі складові: короткий опис; передумови; основний потік подій; альтернативний потік подій; постумови. У таблиці 3.1 наведене опис варіантів використання з діаграми на рис.3.1.

Таблиця 3.1 - Опис варіантів використання системи

Опис	Складова
1	2
Тестування конфігурації	
Короткий опис	виконання тестування швидкості сайту, тестування конфігурації веб-додатка, тестування продуктивності сторінок сайту і тестування на масштабування

Продовдення табл. 3.1

1	2
Передумови	Система запущена та виводить початкову форму програми.
Основний потік подій	1. Тестувальник запускає процес тестування веб-додатка
Альтернативний потік подій	Немає
Постумови	засобами автоматизації тестування формуються результати тестування
Тестування серверної частини	
Короткий опис	проведення тестування навантаження серверної частини додатка 1С8
Передумови	прецедент починається за ініціативою Тестувальника
Основний потік подій	Тестувальник запускає процес тестування навантаження серверної частини додатка
Альтернативний потік подій	Немає
Постумови	засобами програми формуються результати тестування

3.1.3 Побудова концептуальної моделі даних

Концептуальна модель — модель предметної області, що складається з переліку взаємопов 'язаних зрозуміти, що використовуються для опису цієї області, разом з властивостями й характеристиками, класифікацією цих зрозуміти, за типами, ситуацій, ознаками в даній області і законів протікання процесів в ній.

Концептуальна (змістовна) модель — це абстрактна модель, що визначає структуру модельованої системи, властивості її елементів і причинно-наслідкові зв'язки, властиві системі і суттєві для досягнення мети моделювання.

Основні елементи концептуальної моделі:

- умови функціонування об' єкта, визначені характером взаємодії між об' єктом і його оточенням, а також між елементами об' єкта;
- мета дослідження об' єкта та напрямок покращення його функціонування;
- можливості керування об' єктом, визначення складу керованих змінних об' єкта.

Діаграма «сутності - зв'язку» виступає засобом опису схеми бази даних на концептуальному рівні проектування. На діаграмах концептуального рівня сутності зображуються прямокутниками, атрибути - еліпсами, зв'язки - ромбами.

Однак дана програмна розробка не є програмним забезпеченням подібного типу структури та не потребує функціональності бази даних .

3.1.4 Розробка діаграм діяльності

Діаграма діяльності — в UML, візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходь відбуваються по завершенню дій.

Діаграма активностей може вважатись формою блок-схеми. На рисуннку 3.2 зображено сценарій варіантів використання для роботи розроблюваного програмного забезпечення.

Рисунок 3.2 - Діаграма діяльності моделі хмарного тестування застосунку 1С8

Наведена діаграма сценаріїв діяльності для прецедентів типовий показує послідовність дій та події, що ініціюють дії або є їх кінцевим результатом.

3.1.5 Розробка прототипу інтерфейсу користувача

Робота з програмним забезпеченням відбувається за допомогою інтерфейсу користувача. Беручи до уваги вимоги, що зазначені у технічному завданні, визначимо зовнішній інтерфейс ПЗ.

На рисунках 3.3 – 3.4 представлені прототипи користувацького інтерфейсу програмного забезпечення.

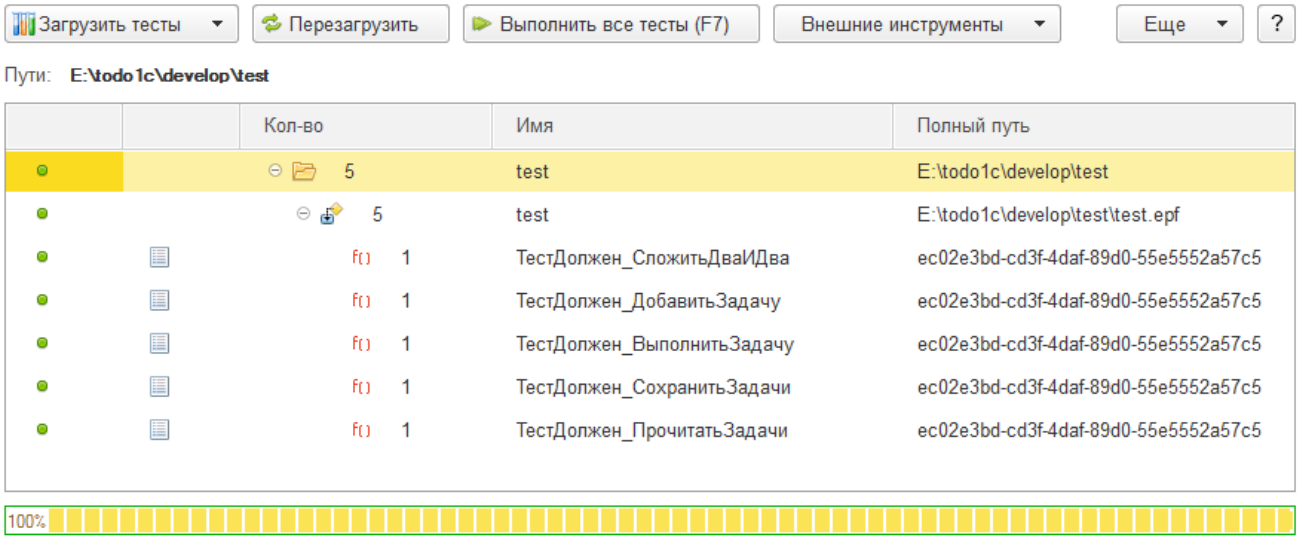


Рисунок 3.3 - Результат тестування простої обробки

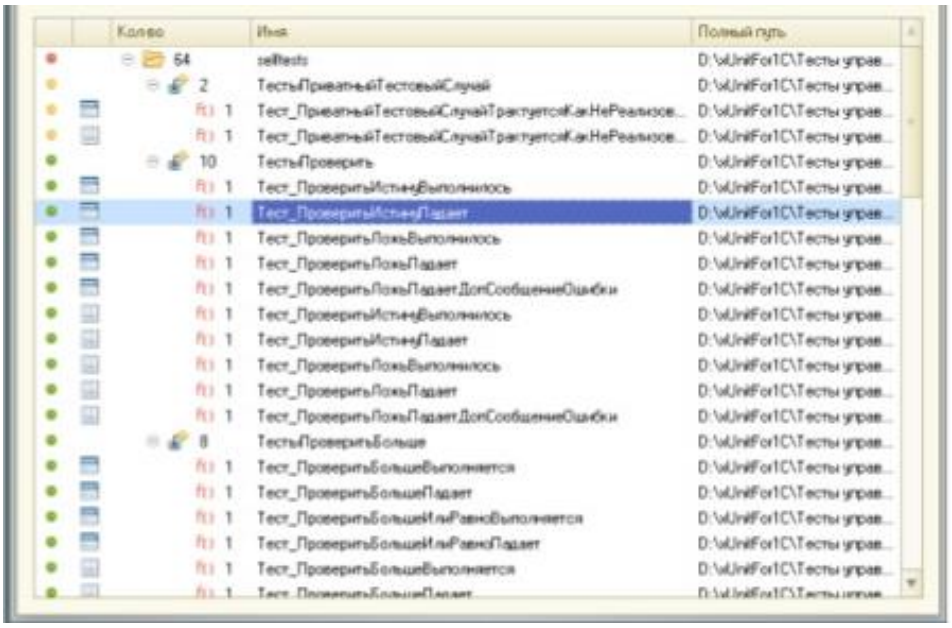


Рисунок 3.4 – Результат перевірки керованого додатку

Для коректної роботи обробки тестування 1С, необхідно обрати файли для тестування та натиснути кнопку «Почати тест». Програма виконує комплексне тестування в залежності від формату вхідного файлу.

3.2 Технічний проект програмного забезпечення

3.2.1 Розробка статичної моделі програмного забезпечення

Діаграма класів — статичне представлення структури моделі, відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів служити для представлення статичної структури моделі системи в термінології класів об' єктно — орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об' єкти й кооперації, а також їхні відносини [29].

Діаграма класів програмного забезпечення для автоматизації роботи розроблюваної програми представлена на малюнку 3.5. На діаграмі класів визначені класи для перевірки розміру коду, його виконуваності (математичні операції), функціональне та навантажувальне тестування. Вказано атрибути та їх типи, а також методи.

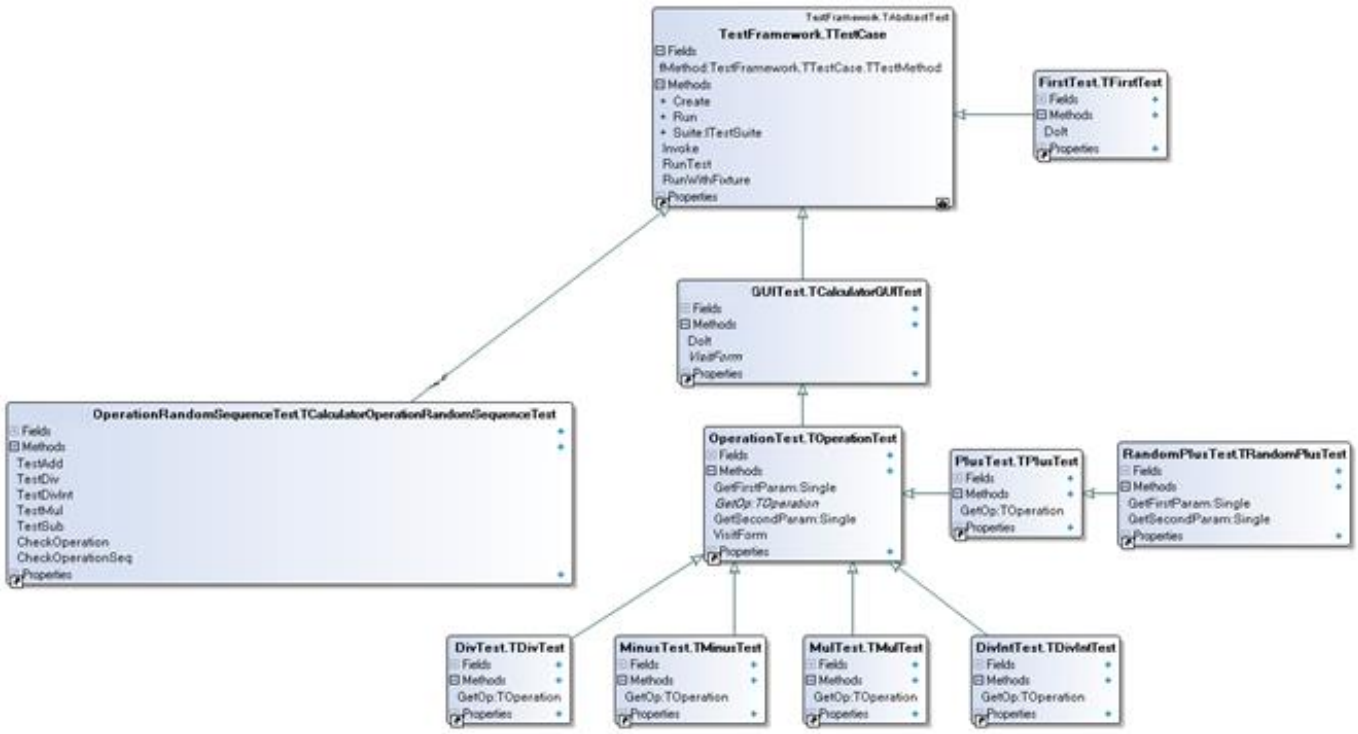


Рисунок 3.5 - Діаграма класів програмного забезпечення

3.2.2 Побудова динамічної моделі програмного забезпечення

Динамічна модель програми представлена у вигляді діаграми послідовності. В магістерській роботі представлені діаграми послідовності за деякими варіантами використання.

Діаграма послідовності для варіанту використання «Тестування конфігурації» представлена на рисунку 3.6.

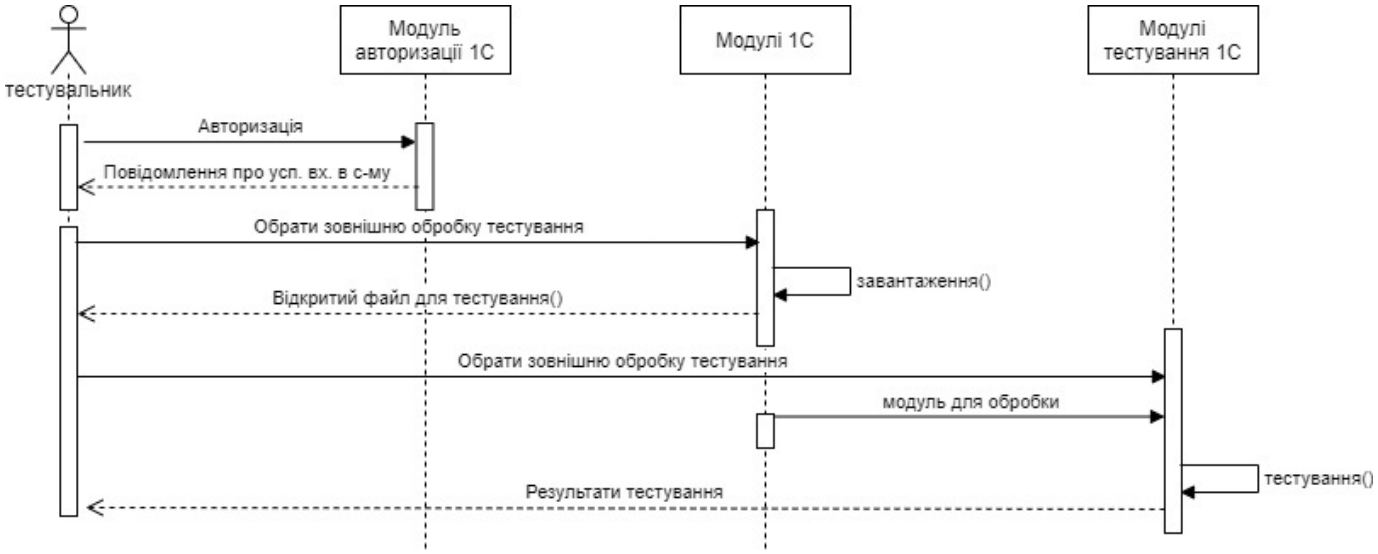


Рисунок 3.5 - Діаграма послідовності для тестування конфігурації 1С

3.3 Робочий проект програмного забезпечення

3.3.1 Обгрунтування вибору мови програмування

Для розробки програмного забезпечення використовується ПЗ 1С: Підприємство.

1С: Підприємство 8 – програма призначена для ведення бухгалтерського, управлінського, фінансового обліку на підприємстві та управління всіма аспектами його діяльності.

Особливістю системи є ті що вона в основній своїй масі експлуатується на підприємствах пост радянського простору і має адекватну для свого ринку вартість, європейські системи такого класу як правило коштують у кілька разів дорожче.

Незважаючи на невисоку вартість за всіма параметрами програма має конкурентоспроможну функціональність і переваги за рахунок наявності в системі модулів бухгалтерського і податкового обліку які розроблені для всіх країн в яких вона використовується.

Завдяки практично монопольному становищу на ринку програма 1С :

Підприємство має якісну технічну та методичну підтримку від фірм партнерів яких в Україні налічується близько 500. Розробники програми 1С: Підприємство контролюють якість послуг, що надаються фірмами партнерами, проводять заходь для їх навчання і сертифікації з метою підвищення якості послуг і максимального задоволення бажань замовників.

Для користувачів які хочуть впроваджувати програми 1С : Підприємство самостійно існує величезна кількість методичної літератури російською та українською мовою, велика кількість різноманітних навчальних курсів та відеоматеріалів, також є можливість отримати деякі версії програм для навчання і експлуатації безкоштовно.

Широке поширення програма отримала починаючи з версії 1С: Підприємство 7.7, вона так сподобалася споживачеві що незважаючи на появу конфігурацій під 1С : Підприємство 8 для України ще в 2008 році, в 2017 році її усе ще використовують близько 10% українських компаній від загального ринку програм 1С: Підприємство в Україні .

Потокова версія програми 1С: Підприємство 8.3, при її створенні розробники особливу увагу приділяли технологіям для експлуатації програми засобами мережі Інтернет і тепер крім роботи в режимі розподіленої інформаційної бази (РІБ) у версії 8.3 з'явилася можливість повноцінної роботи програм 1С : Підприємство в режимі Тонкого клієнта і Web клієнта.

Система 1С : Підприємство 8 добрі підходить для споживачів різного розміру бізнесу.

Програма 1С: Підприємство 8 може працювати як у файловому режимі бази даних так і Клієнт -серверному варіанті, для компаній які використовують 10 і більше користувачів краще використовувати Клієнт -серверний варіант програми. Для організації роботи програми в Клієнт -серверному варіанті крім ліцензій на Основну програму і додаткові робочі місця необхідно придбати ліцензію на Сервер 1С : Підприємство 8 і вибрати один з декількох варіантів платних або безкоштовних SQL

серверів, платні: Microsoft SQL Server, Oracle Database, безкоштовні PostgreSQL, IBM DB2. Робота в Клієнт -серверному варіанті забезпечує більш швидкий доступ до даних і кращий захист інформації в порівнянні з файловим варіантом.

Система 1С : Підприємство 8 підтримує роботові з широким спектром торгового обладнання : фіскальні реєстратори, вага, ПЗСТ термінали, сканери штрих-кодів, датчики обліку відвідувачів.

Програма 1С: Підприємство 8 складається з двох компонентів Середовище розробки (Конфігуратор) і Конфігурації (прикладні рішення). [30]

3.3.2 Тестування та випробування програмного забезпечення

Найбільшої популярності набули дві стратегії тестування програм.

Стратегія «чорного ящика», є тестуванням з управлінням по даним, або тестуванням з управлінням по входу-виходу. При використанні цієї стратегії програма розглядається як чорний ящик. Таке тестування має на меті з'ясування обставин, у яких поводження програми не відповідає її специфікації.

Тестові ж дані використаються тільки у відповідності зі специфікацією програми (тобто без урахування знань про її внутрішню структуру). При такому підході виявлення всіх помилок у програмі є критерієм вичерпного вхідного тестування. Останнє може бути досягнуте, якщо в якості тестових наборів використовувати всі можливі набори вхідних даних.

Стратегія «білого ящика», або стратегія тестування, що управляє логікою програми, дозволяє досліджувати внутрішню структуру програми. У цьому випадку особа одержує тестові дані шляхом аналізу логіки програми. Цей метод характеризується ступенем, у який тесті виконують або покривають логіку (вихідний текст) програми.

Тестування було проведено за моделлю Apdex на конфігурації Управлінні Торгівлею 11.3, фрагмент якого наведено у лістингу 3.1

Лістинг 3.1 Фрагмент тестування ПЗ

```
#Если Сервер ИЛИ ТолстыйКлиентОбычноеПриложение ИЛИ ВнешнееСоединение Тогда
#Областъ Обработчики
//Выполняет проверку конфигурации в соответствии с переданным списком требований.
Процедура 0бра5отхаПроведения(Отказ, РежимПроведения)
          Проверка. 0бработкаПроведенияДокументаПроверки(ЭтотОбъект).
КонецПроцедуры
// Обработчик перед записью документа. Синхронизирует информацию конфигурации по
версии.
Процедура ПередЗаписью(Отказ, РежимЗаписи, РежимПроведения)
          Если ОбменДанными.Загрузка Тогда Возврат;
          КонецЕсли
     Если Версия.Владелец то Конфигурация Тогда
     Конфигурация = Версия.Владелец;
          КонецЕсли;
КонецПроцедуры

#КонецОбласти
```

Під час випробувань було проведено повне функціональне тестування, також навантажувальне тестування і тестування на відмову всього ПЗ. Випробування програми було виконано згідно Додатку Д - «Програма та методика випробувань».

Випробування ПЗ проводилося на цільовому обладнанні (комп' ютері) в тій конфігурації, що заявлена в Додатку А - Технічне завдання. Всі виявлені недоліки ПЗ булі зафіксовані та усунуті розробником до моменту впровадження ПЗ в дію. Випробування було проведене за стратегією написання юніт-тестів. У повному обсязі дану інформацію і результати тестування можна у Додатку Д - «Програма та методика випробувань»

Також, у процесі тестування була перевірена функціональність програмного забезпечення хмарного тестування файлів 1С та розробка програми для реалізації. У таблиці 3.2, що наведена нижче, вказано перелік випробувань основних функціональних можливостей.

Таблиця 3.2 – Випробування ПЗ

№	Дія	Очікуваний результат	Результат перевірки	Зауваження
1	2	3	4	5
1	Випробування на імпорт непідтримуваного формату файлу	Виводиться повідомлення про помилку вхідного файлу	Виконано	
2	Випробування на імпорт коректного формату файлу	У результаті файл успішно завантажився та у вікні було виведено його назву.	Виконано	
3	Випробування на тестування параметрів без завантаження файлу	Вікно з результатами порожнє порожнє. Програма очікує імпорту файлу для тестування	Виконано	
4	Випробування на тестування порожнього файлу зовнішньої обробки	Помилка зчитування файлу зовнішньої обробки	Виконано	
6	Випробування на експорт до файлу csv без проведення тестування	Повідомлення про відсутність данних для відвантаження	Виконано	
7	Випробування на експорт даних до файлу csv після успішного тестування	Випливає діалогове вікно з пропозицією оберни каталог для збереження файлу з даними csv	Виконано	
8	Випробування на правиьність очищення полів з даними	Вікна та поля очищені. Імпорт наступного файлу для тестування успішно прийнято	Виконано	

4 РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ХМАРНОГО ТЕСТУВАННЯ ЗАСТОСУНКІВ 1С

Практичним результатом кваліфікаційної роботи є програма для хмарного тестування застосунків 1С. В ході дипломного проектування було здійснено постановку задачі на розробку ПЗ, виконано ескізний, технічний та робочий проекти ПЗ.

Згідно з вимогами до ПЗ, наведеними в технічному завданні (Додаток А), програмне забезпечення надає користовачутакі функції:

- імпорту зовнішніх файлів з даними (кодом модулю);
- тестування файлу з даними;
- виконання навантажувального тестування на апаратну частину відділеного сервера;
- тестування програмного коду на «масштабованість».

Програмне забезпечення повністю відповідає вимогам до організації вхідних та вихідних даних, вимогам до надійності та іншим вимогам, зазначеним в технічному завданні (Додаток А).

Також була розроблена наступна програмна документація:

− технічне завдання (Додаток А);

− опис програми (Додаток В)

− текст програми (Додаток Б);

− інструкція користувача (Додаток Г);

− програма та методика випробувань ПЗ (Додаток Д).

Програмне забезпечення було розроблено на вбудованій мові програмування технологічної платформи 1С та в середовищі розробки 1С: Підприємство 8.

Створене ПЗ було протестоване на відповідність технічному завданню.

Під час тестування ПЗ показало повну працездатність.

Розроблене ПЗ може бути використано користувачем, не здатним до програмування. Інтерфейс є інтуїтивно зрозумілим.

Програмне забезпечення для хмарного тестування застосунків 1С призначене для використання на персональних комп'ютерах, що працюють під управлінням наступних операційних систем: ОС MS Windows XP/7/8/8.1/10.

5 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ

Найбільш важливим моментом при проектуванні програмного забезпечення для розробника, з економічної точки зору, є процес формування її майбутньої вартості. Створення програмного забезпечення вимагає одноразових витрат на її розробку, придбання необхідних технічних засобів, а також поточних витрат на функціонування продукту. Економія від функціонування програмного продукту визначається з урахуванням витрат на його експлуатацію. Відношення цієї економії до витрат на створення програмного продукту характеризує економічну ефективність капітальних вкладень.

Економічні показники визначаються по діючим на момент розрахунку оптовим цінам, тарифам і ставкам заробітної плати. У зв'язку з актуальністю даного питання для студентів, що навчаються за напрямком «Програмне забезпечення автоматизованих систем» та із заподій відсутності інформаційного забезпечення студентів з даної дисципліни, у рамках даної магістерської роботи. Економічні показники визначаються діючим на момент розрахунку оптовим цінам, тарифам і ставкам заробітної плати.

5.1 Розрахунок витрат на створення і експлуатацію програмного забезпечення

Витрати на розробку продукту складаються з витрат на заробітну платню розробника, на збірку (або закупівлю) комп'ютера для програмування, на експлуатацію ЕОМ, на засоби розробки та витрати на матеріали.

Розробка програмного забезпечення виконується програмістом, місячний оклад якого складає приблизно 8000 грн. Додаткова заробітна плата складає 20% від окладу.

Виходячи з вищевказаного, основна і додаткова заробітна плата розробника складає 9600 грн./міс, а вартість сучасного ПК - 13200 грн. (приведена вартість на базі Intel Core i5 - 10400). При вартості кіловат -години електроенергії рівної 1.68 грн. (згідно діючого тарифу)[41], розраховується вартість розробки програми. Витрати на допоміжні матеріали приведені в таблиці. 5.1.

Таблиця 5.1 - Витрати на допоміжні матеріали

Пункти витрат	**Торба (грн.)**
Папір А4 (500 арк, упаковка, клас А)	120
Заправлення картріджа до принтера	160
CD -диск	15
Непередбачені витрати	300
Разом матеріали і комплектуючі	595

Вартість програми оцінювання розміру веб- застосунків на базі Django - фреймворку розрахуємо за наступною формулою:

$$Спр = (Ззп + Зсз + Ззг + Зе) * Т + Зм, \quad (5.1)$$

де

Т - тривалість розробки (міс);

Ззп - основна і додаткова заробітна плата, грн. :

Зсз - відрахування на соціальні заходь (38% від основної і додаткової заробітної плати, грн.);

Ззг - загальногосподарські виграти (10% від основної заробітної плати, грн.);

Зе - витрати на електроенергію;

Зм - витрати на основні і допоміжні матеріали;

Зе при споживанні потужності 0,5 кВт, тривалості роботи за місяць, рівної 21*8=168 рік., вартості кіловат -години електроенергії 1,68 грн. складає :

$$Зе = 168 * 1,68 * 0,5 = 141,12 грн.$$

Всі витрати на розробку ПЗ можна побачити в таблиці 5.2.

Таблиця 5.2 - Витрати на розробку ПЗ.

Найменування витрат	Одиниця	Кількість
Тривалість розробки	міс.	1,5
Основна і додаткова заробітна	грн.	9600
Відрахування на соціальні заходь	грн.	3072
Загальногосподарські витрати	грн.	800
Витрати на основні і допоміжні матеріали	грн.	595
Витрати на електроенергію	грн.	141,12

Відповідно вартість програми:

$$Спр = (8000 + 3072 + 800 + 141,12) * 1,5 + 595 = 18614,68 \text{ грн.}$$

Амортизаційні відрахування на устаткування складають 60% балансової вартості в рік:

$$Аоб = 13200 * 0,6 = 7920 \text{ грн.}$$

У масштабах підприємства річні витрати на основні і допоміжні матеріали (гнучкі диски, папір) визначаються в розмірі 5% вартості основного устаткування :

$$Вм = 13200 * 0,05 = 660 \text{ грн.}$$

Річний обсяг робіт ПК у годинах визначається в такий спосіб:

$$Фм = 264,5 * Тз$$

де *Тз* - середнє місячне завантаження устаткування (близько 4 годин);

264,5 – середня кількість робочих днів у році.

Отже, річний обсяг роботи ПК складі:

$$Фм = 264,5 * 4 = 1058 \text{ годин}$$

Витрати електроенергію *Зе* при 1058 годинах роботи устаткування в рік складуть

$$Зе = 1058*0,5*1,68 = 888,72 \text{ грн.}$$

Експлуатаційні витрати для ПК за рік складатимуть:

$$Взр = 7920+660+888,72 = 9468,72 \text{ грн.}$$

Отже, у перший рік витрати на створення й експлуатацію програми складуть :

$$Зсе = 18614,68 + 9468,72 = 28083,40 \text{ грн.}$$

5.2 Економічна ефективність розробки і впровадження програмного забезпечення

Основним показником економічної ефективності функціонування програмного продукту є підвищення ефективності керування інформацією у вигляді зниження витрат на керування при одночасному збільшенні швидкості і якості одержання потрібного результату.

Крім багатьох інших негативних ефектів, ручна обробка інформації спричиняє наступні негативні економічні ефекти:

— високі витрати на складування паперових документів (сейфи, шафи, теки й ін.);

— підвищені витрати на канцтовари;

— витрати, пов' язані з роботою виявлення раніше допущених помилок (людський чинник).

До числа основних факторів, що визначають приріст прибутку в зв'язку з упровадженням програми, відносяться:

— підвищення продуктивності праці;

— вивільнення робочого годині.

Крім того, не піддається прямій грошовій оцінці підвищення оперативності керування, якість одержуваних результатів, поліпшення організації праці і т.д.

Обов'язковою умовою визначення економічної ефективності програми є порівнянність усіх показників у часі, за цінами й іншими нормами, використовуваними для визначення показників, за змістом і кільком елементів витрат.

Визначимо пряму економічну ефективність, грунтуючись на тому, що впровадження програмного продукту вивільняє 0,6 працівника (за експертною оцінкою фахівців підприємства).

Зарплата 0,6 працівника в рік складає:

*8000 * 12 * 0,6 = 57600,00 грн.*

Річний економічний ефект розраховується по формулі:

$$Эгод = \triangle Сп - Еп * k$$

де ∆Сп - вивільнені кошти після впровадження програмного продукту 57600,00 грн.) мінус експлуатаційні витрати (9468,72 грн.);

Еп - коефіцієнт ефективності (дорівнює коефіцієнту амортизації - 0,6);

k - одноразові витрати на впровадження продукту (*18614,68 грн.).*

$$Эгод = 57600,00 - 9468,72 - (0,6 * 18614,68) = 36942,47 \text{ грн.}$$

Рядків окупності програми розраховується по формулі:

$$T = k\Delta Сп = 18614,68\ 48131,28 = 0.39 \text{ (року)} \approx 5 \text{ (місяців)}$$

Отже рядків окупності програмного продукту складає приблизно 5 місяців.

Проаналізувавши результати можна зробити висновок про доцільність розробки. У даній роботі запропоновано використовувати методологію, що базується на розробленій моделі якості, специфічній для даного виду ПЗ.

Витрати на перший рік на створення й експлуатацію програми складуть : *28083,40 грн.* Рядків окупності програмного продукту складає приблизно 5 місяців.

6 ОХОРОНА ПРАЦІ

На великих підприємствах станом на сьогодні діє система заходів, що спрямована на запобігання та попередження виникнення різних ситуацій, які можуть призвести до загрози життю і здоров'ю працівника. Дана система заходів називається охороною праці.

Під охороною праці розуміється сукупність способів, засобів і дій, спрямованих на скорочення у рамках підприємств або галузей травматизму, ситуацій, які можуть надати шкоду здоров'ю простого робітника.

Законодавство по охороні праці складається з дійсних Законів України «Про охорону праці», «Про охорону здоров'я», «Про пожежну безпеку», «Про забезпечення санітарного та епідеміологічного благополуччя населення», Кодексу законів про працю України та інших нормативних актів.

У випадку, коли міжнародними договорами чи догодами, у яких бере доля Україна, установлені більш високі вимоги до охорони праці, чим ті, котрі передбачені законодавством України, застосовуються правила міжнародного договору чи попади.

Визначення терміну «Охорона праці» наведене в першій статті закону України «Про охорону праці». Охорона праці - це система правових, соціально - економічних, організаційно - технічних, санітарно - гігієнічних і лікувально - профілактичних заходів і заходів вкладених у збереження життя, здоров'я та перемоги працездатності людини під година праці. Закон України визначає основні напрямки по реалізації конституційного права на охорону життя і здоров'я в процесі трудової діяльності, регулюють при участі відповідних державних органів відносини між власником підприємства, чи встанови організації, чи органом і роботодавцем з питань зовнішнього середовища і встановлює єдиний порядок організації охорони праці в Україні [31].

6.1 Аналіз небезпечних і шкідливих факторів у офісному приміщенні з персональними комп'ютерами

Цілком безпечних і нешкідливих виробництв не існує. Головна завдання охорони праці - звести до мінімальної ймовірність ураження або захворювання працівника, а також виявлення і вивчення шкідливих факторів, їхній вплив на людину і навколишнє середовище.

Небезпечним виробничим чинником називається такий виробничий чинник, вплив котрого на працюючого у визначених умовах призведе до травми або до іншого раптового, різкого погіршення самопочуття. Прикладами небезпечних факторів можуть служити відкриті струмоведучі частини устаткування, що рухають деталі машин та інше. Прикладами шкідливих факторів являються шкідливі домішки в повітрі, несприятливі метеорологічні умови, шум, вібрації, недостатнє освітлення. [32]

Рівень безпеки є результатом взаємодії людини і того середовища, системи безпеки, що діє на виробництві. До факторів, які впливають на виконання виробничого процесу відносять мікроклімат приміщення, шум та вібрацію, освітлення, електробезпеку, пожежну безпеку, пив та інші.

Шумом є всякий небажаний для людини звук. Шум на виробництві завдає великої шкоди, шкідливо впливаючи на організм людини і знижуючи продуктивність праці. Шум навіть коли невеликий (при рівні 50-60 дБа), створює значне навантаження на нервову систему людини, роблячи на нього психологічний вплив. Під впливом шуму, що перевищує 85-90 дБа, у деру чергу знижується слухова чутливість на високих частотах.

Нормою виробничого шуму є рівень звуку до 85 дБ. Якщо рівень перешкод становить 20 дБ, то такий шум не заважає розбірливості мови. З підвищенням рівня перешкод до 70 дБ та вище мова стає нерозбірливою.

Джерелами шуму найчастіше є : виробниче устаткування для випікання, перемелювальні пристрої, двигуни, пневматичні та електричні інструменти, верстати, будівельна техніка худе.

Небезпечним чинником для роботи програміста є також робота за комп'ютером. Найбільшому ризику піддаються зорова, опорно-рухова та нервово-психічна системи. Монітор випускає випромінювання декількох видів: рентгенівське, ультрафіолетове, інфрачервоне, електромагнітне. Для шкірного з цих випромінювань розроблені гранично допустимі норми. Норми передбачають, що опроміненню піддається верхня частина тулуба. Згадані норми встановлені з розрахунку на кожен вид опромінення в окремо, хоча реально всі поля діють одночасно, а їх комплексний вплив досі не досліджено.

Відеоапаратура порушує рівновагу між позитивно і негативно зарядженими іонами в повітрі. Електростатичне поле дисплея притягає негативні іони, порушуючи загальний баланс атмосфери. Це також шкодити здоров'ю. Вже через годину роботи біля монітора спостерігається майже повне зникнення негативних іонів. Вісь чому необхідно, щоб до робочого місця за комп'ютером проникало свіже повітря.

Освітленість робочої зони комп'ютера повинна відповідати стандартам та бути вище яскравості монітора. Відстань від очей до екрана повинна бути не менше ніж півметра. Висота крісла має бути такою, щоб очі булі на одному рівні з центром монітора. Фахівці підтверджують, що саме очі найбільш страждають при роботі з комп'ютером. Занадто відсунутий монітор призводить до перенавантаження м' язів голови та шиї, через що тиск на їх роботові зростає приблизно в три вражай, судини шиї стискаються, погіршуючи кровопостачання до голови. Крім того, людині, що сидить у такій незручній позі, доводитися щоразу відкидати голову назад, для фіксування уваги на ближчих об' єктах. Це посилює вигин шийного відділу хребта. Через деякий годину це може призвести до головної болі та болю у кінцівках, оскільки нерви, що відходять від спинного мозку в області шиї, простягаються до кінчиків пальців.

Малорухома та тривала сидяча поза за комп'ютером шкідлива для опорно-рухового апарату, що веде до застою крові в органах людини. Це особливо проявляється при фізіологічному положенні різних частин тіла і тривало повторюваних одноманітних рухах. Під година роботи за комп'ютером людина сидить кілька годин поспіль в незручному становищі. Це не тільки загрожує втомою і загальним знесиленням організму, а й може призвести до розвитку остеохондрозу різних ділянок хребта - шийного, грудного, попереково-крижового.

Суттєвий вплив на стан організму ІТ - спеціаліста та його працездатність здійснює мікроклімат (метеорологічні умови) у відділі, під яким розуміють клімат внутрішнього середовища цих приміщень, що визначається діючою на організм людини сукупністю температури, вологи, руху повітря та теплового випромінювання нагрітих поверхонь. Несприятливі метеорологічні умови приводять до швидкої втоми працівника, частішу його захворюваністьі та зниження продуктивності праці.

Світло впливає не лише на функції органів зору, а й на діяльність організму в цілому. При поганому освітленні людина швидко втомлюється, працює менш продуктивно, зростає потенційна небезпека помилкових дій і нещасних випадків. Для створення оптимальних умів зорової роботи слід враховувати не лише кількість та якість освітлення, а й кольорове оточення. Так, при світлому пофарбуванні інтер'єру завдяки збільшенню кількості відбитого світла рівень освітленості підвищується на 20-40%, різкість тіней зменшується, покращується рівномірність освітлення.

При надмірній яскравості джерел світла та оточуючих предметів може відбутися засліплення працівника. Нерівномірність освітлення та неоднакова яскравість оточуючих предметів призводять до частої переадаптації очей під година виконання роботи, і як наслідок цього - до швидкого їх втомлення. Тому поверхні, що добрі освітлюються і знаходяться в полі зору, краще фарбувати в кольори середньої світлості.

При освітленні ІТ - відділу використовують природне освітлення, що створюється світлом неба, штучне, здійснюване електричними лампами, і сполучене,

при якому у світлий година доби недостатній за нормами, природне освітлення доповнюється штучним. [32]

Для забезпечення здоров'я працюючих приймаються необхідні запобіжні міри захисту, що повністю або частково ізолюють доступ до зони, в якій діють шкідливі фактори, та виключають їх дію в разі проникнення людини у простір, де сморід виникають. [32]

Велика кількість шуму негативно впливає на розумову роботові мозку, значно знижує продуктивність роботи програміста.

До методів боротьби із шумом відносяться :

- зменшення шуму в джерелі виникнення;
- зміна напрямку випромінювання шуму;
- акустична обробка приміщень;
- установка звукоізолюючих огороджень, кожухи, екрани, кабіни;
- установка глушителів шуму;
- використання засобів індивідуального захисту (вкладиші, навушники, шоломи).

Конструктивно сморід можуть бути зроблені у вигляді гратчатих, сітчастих та непрозорих перешкод із металу, деревини худе. Віброізоляція зменшує рівні вібрації, що передаються від джерела на тіло працюючого. Вона здійснюється введенням між джерелом вібрації та працюючим проміжного пружного зв'язку.

Заходь, щодо зниження негативного впливу мікроклімату:

1. впровадження раціональних технологічних процесів;
2. механізація та автоматизація виробничих процесів;
3. захист працівників різними типами екранів;
5. раціональна теплова ізоляція устаткування;
6. раціональне розміщення устаткування;
7. раціональне планування та конструкторське рішення виробничих приміщень;

8. раціональні вентиляція та опалювання;

Для збереження здоров'я очей, та запобіганню перевтоми, робоче приміщення спеціаліста ІТ - відділу повинне відповідати наступним вимогам:

- створювати на робочій поверхні освітленість, що відповідає характеру зорової роботи і не є нижчою за встановлені норми;

- не повинне чинити засліплюючі дії як від самих джерел освітлення, так і від інших предметів, що знаходяться в полі зору;

- забезпечити достатню рівномірність освітлення;

- не створювати на робочій поверхні тіней;

- повинне бути надійним в експлуатації, економічним та естетичним.

Робоче місце має відповідати сучасним вимогам ергономіки і забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання (дисплея, клавіатури, принтера) і документів :

1. Висота робочої поверхні робочого столу має регулюватися в межах 680-800 мм, а ширина і глибина - забезпечувати можливість виконання операцій у зоні досяжності моторного поля (рекомендовані розміри: 600-1 400мм, глибина - 800-1 000мм).

2. Робочий стіл повинний мати простір для ніг заввишки не менше ніж 600мм, завширшки не менше ніж 500мм, завглибшки (на рівні колін) не менше ніж 450мм, на рівні простягнутої ноги не менше ніж 650мм. Робочий стілець має бути підйомно -поворотним, регульованим за висотою, з кутом і нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння поверхня сидіння має бути плоскою, передній край - заокругленим.

3. Робочі місця слід розташовувати відносно світових прорізів так, щоб природне світло падало переважно з лівого боку.

4. Монітор має розташовуватися на оптимальній відстані від очей користувача, що становить 600-700мм, але не ближче ніж за 600мм з урахуванням розміру літерно -цифрових знаків і символів. Розташування екрана монітору має

забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30 градусів до нормальної лінії погляду працівника.

5. Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5-15 градусів.

6. Для забезпечення захисту і досягнення нормованих рівнів комп'ютерних випромінювань необхідно застосування приекранних фільтрів, локальних світлофільтрів та інших засобів захисту, що пройшли випробування в акредитованих лабораторіях і мають гігієнічний сертифікат.

На даний годину, це поки що найреальніша можливість захистити людину, що працює біля комп'ютера від професійного захворювання.

Електрика широко застосовується у всіх галузях. Тому питанню електробезпечності необхідно приділяти велику увагу. Електробезпечність - система організаційних і технічних заходів та засобів, що забезпечують захист людей від шкідливого і небезпечного впливу електричного струму, електричної дуги, та інше. Проходячи через організм, електричний струм робить термічні, електролітичні і біологічні дії. Це різноманіття дій електричного струму нерідко приводити до різних електричних травм, що умовно можна звести до двох видів: місцеві електричні травми та загальні електричні травми.

Основні поразки струмом наступні:

- випадковий дотик або наближення на небезпечну відстань до струмопровідних частин, що знаходиться під напругою;

- поява напруги на металевих корпусних частинах електроустаткування;

- поява напруги на відключених струмоведучих частинах, на яких працюють люди, унаслідок помилкового включення установки;

Основними заходами захисту від поразки струмом є : забезпечення неприступності струмоведучих частин, що знаходяться під напругою, для випадкового дотику; електричний поділ мережі; усунення небезпеки ураження з появою напруги на корпусах, кожухах і інших частинах електроустаткування, що досягається застосуванням малих напруг, використанням подвійної ізоляції, виявленням потенціалу, захисним заземленням, занулюванням, захисним відключенням і інше; застосування спеціальних електрозахисних засобів - переносних приладів і пристосувань; організація безпечної експлуатації електроустановок.

Занулюванням називається навмисне електричне з' єднання з нулем захисним провідником металевих не струмоведучих частин, що можуть виявитися під напругою. Призначення нульового захисного провідника - створення струмові короткого замикання ланцюга з малим опором, щоб цей струм був достатнім для швидкого спрацьовування захисту, тобто швидкого відключення ушкодженої установки від мережі.

Захисне відключення - швидкодіючий захист, забезпечує автоматичне відключення при виникненні в ній небезпеки ураження струмом. Така небезпека може виникнути, зокрема, при замиканні фази на корпус електроустаткування; при зниженні опору ізоляції фаз щодо землі нижче визначеної межі; появі в мережі більш високої напруги; дотик людини до струмоведучої частини, що знаходиться під напругою. Установлюють прилад захисного відключення ручний і автоматичний вимикач. Принцип дії - це швидке відключення від мережі установки, якщо напруга її корпусу щодо землі виявиться вище деякого гранично припустимого значення, унаслідок чого дотик до корпусу стає небезпечним.

Основним нормативним документом, що регламентує вимоги щодо пожежної безпеки є Закон України "Про пожежну безпеку". Цей закон визначає загальні правові, економічні та соціальні основи забезпечення пожежної безпеки на території України, регулює відносини державних органів, юридичних і фізичних осіб у цій галузі незалежно від виду їх діяльності та форм власності. Пожежа - це неконтрольоване

горіння поза спеціальним вогнищем, що розповсюджується в часі і просторі та створює загрозу життю і здоров'ю людей, навколишньому середовищу, призводить до матеріальних збитків. Основними причинами пожежі на підприємстві є:

- небезпечне поводження з увігнемо;

- незадовільний стан електротехнічних пристроїв та порушення правив їх монтажу та експлуатації;

- порушення режимів технологічних процесів;

- невиконання вимог нормативних документів з питань пожежної безпеки.

Власник підприємства, повинний:

- розробляти комплексні заходь щодо забезпечення пожежної безпеки;

- організувати навчання працівників правилам пожежної безпеки та пропаганду заходів щодо їх забезпечення;

- мають бути присутні засоби протипожежної безпеки.

Система протипожежного захисту - це сукупність організаційних заходів, а також технічних засобів, спрямованих на запобігання впливу на людей небезпечних факторів пожежі та обмеження матеріальних збитків від неї. На підприємстві у шкірному цеху та кабінеті є по два вогнегасника.

У результаті проведеного дослідження можна зробити наступні висновки. Під година розумової або фізичної праці людина сприймає комплекс шкідливих виробничих чинників, які можуть позитивно або негативно відзначитись на її здоровий' ї та продуктивності праці. Рівень цих факторів залежить від виду робочої діяльності, але можна сказати, що навіть робота в офісі може нашкодити здоров'ю працівника.

На великих підприємствах шкідливих чинників дуже багато, та їх дія безпосередньо негативно впливає на організм людини в цілому, та на окремі її органи. Постійний вплив цих факторів може призвести до професійних або хронічних захворювань.

Працівникам на потенційно небезпечному виробництві або з небезпечним обладнанням необхідно пам'ятати про засоби індивідуального захисту, правила поводження зі спеціальним обладнанням, адже від цього напряму залежить особістові здоров'я працівника.

Керівництву підприємства необхідно забезпечувати працівників та робітників необхідними засобами індивідуального захисту, вчасно слідкувати за станом робочих місць, що відповідають інструкції з техніки безпеки та не шкодити здоров'ю людини.

6.2 Розрахунок системи штучного освітлення у офісному приміщені з персональними комп'ютерами

У приміщені необхідно організувати змішане освітлення, тобто сполучення природного і штучного освітлення. Природне освітлення створюється бічним освітленням через вікно. Штучне освітлення використовується при недостатньому природному освітлені. У даному приміщені використовується загальне штучне освітлення. Розрахунок його здійснюється по методу світлового потоку з урахуванням потоку, відбитого від стін і стелі.

Довжина приміщення (А) = 5 м, ширина (В) = 4 м, висота (Н) = 2,5 м, висота робочої поверхні (*hp*)= 1 м, для освітлення приймаємо світильник типу УПД, мінімальна освітлюваність лампи розжарювання за нормами Emin=100лк, коефіцієнт відображення стелі ρп =70%, стін ρз = 50%, робочої поверхні ρр=30%. Напруга мережі 210 В.

Розрахуємо відстань від стелі до робочої поверхні:

$$Ho = h - hp = 2.5 - 1 = 1.5 \text{ м}, \quad (6.1)$$

де Н - висота приміщення, м;

hp - висота робочої поверхні, м.

Визначимо відстань від стелі до світильника (hc) :

$$hc = 0.2 * Ho = 0.2 * 1.5 = 0.3 \text{ м} \quad (6.2)$$

Висота підвішування світильника над освітлюваною поверхнею (h) :

$$h = Ho - hc = 1.5 - 0.3 = 1.2 \text{ м} \quad (6.3)$$

Висота підвішування світильника над підлогою (Hn) :

$$Hn = h + hp = 1.2 + 1 = 2.2 \text{ м} \quad (6.4)$$

Для того, щоб досягти найбільшої рівномірності освітлення приймаємо відношення L/h = 1,5. Таким чином відстань між центрами світильників :

$$L = 1.5 * h = 1.5 * 1.2 = 1.8 \text{ м}.$$

Визначимо необхідну кількість світильників:

$$N = S/l2 = 20 / 1.822 = 6.04 \quad (6.5)$$

приймаємо за 6 шт. Знаходимо індекс приміщення:

$$I(h * (A+B)) = 20 / (1.2 * 9) = 1.85.$$

При *i* = 0,57, коефіцієнтах відображення стелі $\rho_п$ =70%, стін $\rho_з$ = 50%, робочої поверхні $\rho_р$=30% для світильника УПД коефіцієнт використання світлового потоку η = 0,28.

Світловий потік однієї лампи, лм:

$$\Phi = \frac{E_{min} \times S \times Z \times K_з}{N \times \eta \times \gamma}, \quad (6.6)$$

де Emin - рівень мінімальної освітленості за нормами, лк;

S - освітлювана площа приміщення, м2 ;

Кз - коефіцієнт запасу;

Z - коефіцієнт мінімальної освітленості;

N - число світильників;

η - коефіцієнт використання світлового потоку ламп, встановлених у світильнику.

Коефіцієнт запасу Кз враховує експлуатаційне зниження освітленості, в порівнянні із запроектованою, внаслідок забруднення, а також зменшення світлового потоку ламп у процесі їх експлуатації. Для нашого приміщення коефіцієнт мінімальної освітленості (Кз) = 1,3.

Коефіцієнт мінімальної освітленості Z дорівнює відношенню середньої освітленості до мінімальної. Він враховує нерівномірність освітленості і залежить в основному від відношення відстаней між світильниками і від їх типів. Значення цього коефіцієнту для ламп розжарювання приймають Z =1,15.

Ф(6* 0.28) = 1334 лм.

За знайденим світловим потоком вибираємо лампу потужністю 200 Вт, що має світловий потік 3200 лм (Г125-135-200) найбільш близький до розрахункового.

При цьому фактична освітленість (лк) дорівнює:

$$E_{ф} = E_{\min} \times \frac{\Phi_{л}}{\Phi_{р}}, \quad (6.7)$$

де *Фл* - світловий потік обраної лампи, лм;

Фр - світловий потік лампи, отриманої розрахунком, лм.

Єф= 75* 3200/1334 = 179,91 лк.

Загальна потужність освітлювальної установки Рз, Вт, визначається за формулою:

$$P_{з} = P_{л} \times N, \quad (6.8)$$

де Рз - потужність обраної лампи, Вт.

Рз = 200 / 6 = 1200 Вт =1,2 кВт.

Загальна потужність освітлювальної установки становитиме при цьому 1200 Вт.

7 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Основними нормативно-правовими актами України, спрямованими на забезпечення екологічної безпеки на її території, є Конституція, Закони "Про охорону навколишнього природного середовища", "Про охорону атмосферного повітря", "Про природно- заповідний фонд України" худо, Земельний, Водний, Лісовий Кодекси, Кодекс України "Про надра", ухвали Кабінету Міністрів України в законодавстві передбачені права та обов'язки природокористувачів.

Закон України "Про охорону навколишнього природного середовища" [46] є основоположним законодавчим актом, визначає поняття екологічної безпеки та заходь щодо їх забезпечення, екологічні вимоги до розміщення, проектування будівництва, реконструкції, введення в дію підприємств та інших об' єктів, про застосування мінеральних доголивши; передбачає заходь щодо охорони навколишнього природного середовища від шкідливого біологічного впливу; шкідливого впливу фізичних факторів та радіоактивного забруднення, від забруднення виробничими, побутовими та іншими відходами.

Даній закон не лише проголошує, але й передбачає систему гарантій екологічної безпеки людини, вносити певну впорядкованість в систему управління в галузі природокористування, він закріплює право громадян України на безпечне для життя навколишнє середовище. Це невід'ємне право реалізується шляхом участі громадян в обговоренні проектів законодавчих актів та інших рішень в галузі охорони навколишнього середовища; участі в розробці та здійсненні заходів щодо охорони природного середовища, раціонального використання природних ресурсів; об' єднання в громадські природоохоронні організації; отримання повної та достовірної інформації про стан навколишнього природного середовища.

Він надає громадянам України право звертатися до суду з позовом на підприємства, встанови щодо відшкодування збитків, заподіяних здоров'ю та майну внаслідок негативного впливу на навколишнє середовище. Метою екології є вивчення

загальних закономірностей впливу антропогенної діяльності на навколишнє середовище, зокрема промисловості, сільського господарства, транспорту, комунального господарства худе.

Охорону навколишнього середовища розглядають зазвичай як комплекс заходів міжнародного, державного, регіонального, локального рівнів, котрі спрямовані на збереження і забезпечення раціонального природокористування, відновлення, охорону та примноження природних ресурсів країни для блага людського суспільства і підтримання біологічної та екологічної рівноваги біосфери. Охорона довкілля включає джерела забруднення та їхній вплив на окремі екосистеми та біосферу з метою запобігання шкідливого впливу.

Найбільш раціональним заходом, спрямованим на зменшення забруднення атмосфери опалювальним обладнанням, є ліквідація пічних систем завдяки розвитку централізованого теплопостачання. При цьому внаслідок підвищення ККД котелень зменшується кількість спалюваного палива, а отже, і забруднення навколишнього середовища. Окрім того, при централізованому теплопостачанні у великих котельнях можливе очищення димових газів перед викидом їх в атмосферу.

Позитивне значення для розвитку не тільки централізації теплопостачання, але і каналізації, а також охорони природи може дати застосування колекторного прокладання інженерних комунікацій, - а саме - глибокого закладення. Таке прокладання здійснюється тунельним способом без зняття рослинного кулі. Пошкодження рослин, порушення існуючих будівель і дорожнього покриття. А також велике значення для покращення екологічного стану повітряного басейну є відмова від використання вугілля в міських котельнях та перехід їх на природний газ.

Одним з потужних джерел забруднення міського повітря є автомобільний транспорт. У зв'язку з цим виникла необхідність розробки ряду заходів, що дозволяють запобігти забрудненню біосфери. Одним з таких заходів перехід автомобілів з бензиновими та дизельними двигунами на електромобілі, що діють від під заряджених на станціях батареях - акумуляторах. Електромобілі мають кілька

переваг: сморід безшумні, бездимні та прості у використанні. Іншим засобом, який сьогодні найчастіше використовується, є встановлення на автомобілях фільтрів чи використання як палива природного газу, котрий в порівнянні з іншими видами палива менше забруднює повітря.

Щодо видалення побутових відходів, то останнім часом в Швеції почали застосовувати пневматичний транспорт для видалення сміття із сміттєпроводів по горизонтальних підземних каналах до станції, що надає послуги декільком будинкам (мікрорайон). У США, Великобританії, Італії та деяких інших країнах застосовується сплав в каналізацію посріблених відходів з квартир, готелів, ресторанів та інших об' єктів. З цією метою біля раковин ставлять механічні подрібнювачі, з котрих подрібнене сміття разом зі стічною водою видаляється в каналізацію, де воно знешкоджується в очисних установках. У нас в країні сміття збирається в контейнери та сміттєприймальні камери.

Для збору та тимчасового схоронності відходів організовують спеціальні майданчики з твердимо покривом, який забезпечує запобігти забрудненню грунту. Періодичність знешкодження накопичених відходів визначається згідно з діючими санітарними нормами та правилами та залежить від середньоденної температури повітря, при якій відбувається розкладання залишків органічних продуктів.

ВИСНОВКИ

Мета, яка була поставлена в кваліфікаційній роботі –підвищення ефективності тестування ПЗ, яке створене із використанням технологічної платформи 1С, шляхом розробки моделі хмарного тестування, що надає користувачеві можливість скоротити витрати в порівнянні із традиційними методами тестування – досягнута в повному обсязі.

Усі завдання, поставлені при підготовці кваліфікаційної роботи, були успішно виконані.

В процесі проведення досліджень одержані наступні результати:

– проведено огляд сучасних технологій хмарного тестування та аналіз існуючих моделей хмарного тестування програмного забезпечення;

– обґрунтована необхідність побудови моделей хмарного тестування застосунків 1С;

– на основі універсальної моделі хмарного тестування програмного забезпечення було запропоновано структурну схему плану хмарного тестування ПЗ 1С та розроблені моделі хмарного тестування застосунків 1С8 та 1С:Бітрікс

– виконана апробація розроблених моделей, яка підтвердила її ефективність, щодо тестування різних видів застосунків 1С;

– було розроблено програмне забезпечення для хмарного тестування застосунків 1С;

– оформлена необхідна програмна документація.

Також в роботі виконані спецрозділи з охорони труда, охорони навколишнього середовища та економіки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Єгорова О.В., Бичок В.П. Програмні засоби для тестування програмного забезпечення. *Молодий вчений*. 2019. № 11 (75). С.680 – 684.

2 Тестування програмного забезпечення та контроль якості. URL: http://www.znannya.org/?view=software-testing (дата звернення 24.09.2020)

3 Cloud testing: URL: https://en.wikipedia.org/wiki/Cloud_testing (дата звернення: 25.09.2020).

4 Types of Cloud Testing URL:https://etutorialsworld.com/2016/04/types-of-cloud-testing/ (дата звернення: 25.09.2020)

5 Роутон Д. Тестування програмного забезпечення в хмарі: як різні хмарні платформи забезпечують тестування додатків до і після розгортання. IBM Corporation, 2013.

6 Google Cloud. URL: https://cloud.google.com/ (дата звернення: 25.09.2020).

7 Amazon Web Services. URL: https://aws.amazon.com/ru/ (дата звернення: 25.09.2020)

8 Microsoft Azure. URL: https://azure.microsoft.com/ru (дата звернення: 27.09.2020)

9 Сравнение услуг облачных провайдеров. URL: http://la.by/blog/sravnenie-uslug-oblachnyh-provayderov-microsoft- azure-aws-ili-google-cloud (дата звернення: 27.09.2020)

10 What Is Automation Testing Pyramid? URL: https://ru.qatestlab.com/resources/knowledge-center/test-automated- pyramid/ (дата звернення: 29.09.2020)

11 SOASTA. URL: https://en.wikipedia.org/wiki/SOASTA (дата звернення: 29.09.2020).

12 Лучшие инструменты для тестирования мобильных приложений. URL: https://geteasyqa.com/ru/blog/best- mobile-testing-tools/ (дата звернення: 30.09.2020).

13 Тестирование облачных сервисов. URL:

https://www.osp.ru/os/2012/05/13016242 (дата звернення: 30.09.2020)

14 Testing as a Service (TaaS). URL: https://searchcloudcomputing.techtarget.com/definition/Testing-as-a-Service-TaaS (дата звернення: 30.09.2020)

15 Software Testing as a Service. (TaaS) URL: https://www.softwaretestingclass.com/software-testing-as-a-service-taas/ (дата звернення: 30.09.2020)

16 Особенности тестирования облачных сервисов. URL: http://getbug.ru/osobennosti-testirovaniya-oblachnyih-servisov/#_SaaS (дата звернення: 30.09.2020)

17 SaaS vs PaaS vs IaaS: What's The Difference and How To Choose. URL: https://www.bmc.com/blogs/saas-vs- paas-vs-iaas-whats-the-difference-and-how-to-choose/ (дата звернення: 30.09.2020)

18 Cloud Testing Tutorial with SaaS and PaaS [Електронний ресурс]. — URL: http://www.professionalqa.com/cloud-testing-tutorial-with-saas-and-paas (дата звернення: 01.10.2020)

19 Apprenda. URL: https://apprenda.com/ (дата звернення: 01.10.2020)

20 Особенности тестирования облачных сервисов. URL: http://getbug.ru/osobennosti-testirovaniya-oblachnyih-servisov/#_SaaS (дата звернення: 01.10.2020)

21 Открытые бенчмарки для нагрузочного тестирования серверов и вебприложений. URL: https://habr.com/ru/company/1cloud/blog/474474/ (дата звернення: 01.10.2020)

22 Хмарне тестування: переваги, проблеми, типи і поради. URL: https://www.facebook.com/notes/start-it-training-center/%D0%BE%D0%B1%D0%BB%D0%B0%D1%87%D0%BD%D0%BE%D0%B5-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-

%D0%BF%D1%80%D0%B5%D0%B8%D0%BC%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B0-%D0%BF%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D1%8B-%D1%82%D0%B8%D0%BF%D1%8B-%D0%B8-%D1%81%D0%BE%D0%B2%D0%B5%D1%82%D1%8B/2299938326743993/ дата звернення: 10.10.2020)

23 Cloud Computing and Testing Cloud based Applications. URL: https://www.360logica.com/blog/cloud-computing-and-testing-cloud-based-applications/ (дата звернення: 10.10.2020).

24 Оценка интегральной производительности системы по методике APDEX. URL: https://its.1c.ru/db/metod8dev#content:5807:hdoc (дата звернення: 10.10.2020).

25 Нагрузочный тест TPC-1C. URL: http://www.gilev.ru/tpc1cgilv/ (дата звернення: 10.10.2020).

26 Организация виртуальной инфраструктуры 1С в Microsoft Azure. URL: https://infostart.ru/public/897231/ (дата звернення: 15.10.2020)

27 Выбор облачного сервиса для работы в 1С. URL: https://efsol.ru/articles/doud-services.html (дата звернення: 15.10.2020)

28 Открытые бенчмарки для нагрузочного тестирования серверов и вебприложений. URL: https://habr.com/ru/company/1cloud/blog/474474/ (дата звернення: 20.10.2020).

29 Самуйлов С.В. Объектно-ориентированное моделирование на основе UML. URL: http://www.iprbookshop.ru/47277.html (дата звернення: 25.11.2020)

30 1С Підприємство. URL: http://1c.ua/ua/ (дата звернення: 30.11.2020)

31 Закони України: «Про охорону праці». URL: http://zakon4.rada.gov.ua/laws (дата звернення: 01.12.2020)

32 Жидецький В.Ц. Охорона праці користувачів комп'ютерів. Львів: Афіша, 2000. 176 с.

ДОДАТОК А - ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Назва програми: «Побудова моделі хмарного тестування програмного забезпечення 1С та розробка програми для її реалізації».

Область застосування програми обмежена програмним середови щем технологічної платформи1С : Підприємство.

1 Підстави для розробки

Підставою для розробки програмного забезпечення є Наказ ректора НУК імені адмірала Макарова № 1037-уч від 26 жовтня 2020 року на затвердження тем кваліфікаційних (магістерських) робіт кафедри ПЗАС.

2 Призначення програми

2.1 Функціональне призначення

Функціональним призначенням програмного забезпечення є автоматизація процесів перевірки і тестування клієнтської та серверної частин програмного середовища 1С.

2.2 Експлуатаційне призначення

Експлуатаційним призначенням програмного забезпечення є спрощення роботи ІТ - компаній при розробці ПЗ та тестування розроблених програмних модулів для середовища 1С: Підприємство.

3 Вимоги до програмного забезпечення

3.1 Вимоги до функціональних характеристик

3.1.1 Вимоги до складу функцій, що виконуються

Програмне забезпечення повинне забезпечувати можливість виконування наступних функцій:

- можливість імпортувати зовнішній файл з даними (кодом модулю);
- можливість протестувати файл з даними;

– можливість виконати навантажувальне тестування на апаратну частину відділеного сервера;

– можливість протестувати програмний код на «масштабованість».

3.1.2 Вимоги до організації вхідних та вихідних даних

Введення вхідних даних здійснюється користувачем у відповідні поля діалогового вікна програмного забезпечення.

Вихідні дані виводиться у діалоговому вікні програмного забезпечення.

3.2 Вимоги до надійності

3.2.1 Вимоги до забезпечення надійності функціонування програми

Вимоги, що забезпечують стійке функціонування програмного забезпечення повинні включати:

– Засоби перевірки введеної інформації (програмними засобами, повинні бути розроблені функції які контролюють введення інформації і не допускають ведення неповної чи невірної інформації, де це потрібно, і надають попередження чи підказки стосовно вірного введення інформації);

– Засоби, що забезпечують надійне збереження інформації (за рахунок використання операційної системи та створення резервних копій).

3.2.2 Відмова виконання програмного забезпечення через некоректні дії користувача

Відмова програмного забезпечення можлива внаслідок невірних дій користувача з операційною системою або середовищем виконання. Задля усунення вказаних відмов потрібно забезпечити стабільне функціонування операційної системи та середовища виконання програмного забезпечення.

3.3 Вимоги до умов експлуатації

Проектоване програмне забезпечення повинне функціонувати при наступних умовах її експлуатації :

– З програмним забезпеченням має працювати користувач, що ознайомлений з керівництвом користувача і має необхідні навички роботи з ПЕОМ в операційних системах Windows 10/8.x/7/XP.

– Умови експлуатації повинні відповідати вимогам до експлуатації апаратної частини ПЕОМ.

– Умови експлуатації носіїв інформації відповідають зазначеним вимогам у супровідній документації.

3.4 Вимоги до апаратного і програмного забезпечення

Система користувача повинна задовольняти вказаним мінімальним вимогам для коректного запуску:

- Процесор: Celeron N2840 2167 Мгц і вище.
- Оперативна пам'ять: мінімум 2 Гб (рекомендується 4 Гб).
- Жорсткий диск: Для нормальної роботи системі потрібно 300 Мб вільного дискового простору.

Для програміста, розробника даного ПЗ, для подальшого вдосконалення системи та проведення тестів пропонується наступна конфігурація:

- Процесор: Intel Core i5 - 10400 2.9GHz
- Материнська плата: на базі чіпсету H410
- Оперативна пам'ять: 8-16 ГБ DDR4 - 2666
- Жорсткий диск: 1TB SATAIII 7200 rpm
- Твердотільний накопичувач (для ОС) : 128 ГБ SATAIII 3D TLC.

3.5 Вимоги до документації

До програмної документації повинні входити наступні документи :

1) Технічне завдання;

2) Текст програми;

3) Опис програми;

4) Інструкція користувача;

5) Програма та методика випробувань.

Мова всієї документації - українська.

3.6 Вимоги до маркування та пакування

Вимоги до упаковки відсутні.

3.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання відсутні.

4 Стадії та етапи розробки

Стадії та етапи розробки наведене в таблиці А.1.

Таблиця А.1 - Стадії та етапи розробки програмного забезпечення

Стадія створення ПЗ	Етапи робот на стадії	Качан	Кінець
Технічне завдання	Визначення призначення розробки та вимог	08.09.2020	12.09.2020
Ескізний проект	Прийняття принципових рішень по структурі ПЗ. Розробка концептуальної моделі даних.	13.09.2020	24.10.2020
Технічний проект	Вибір інструменту мови кодування. Розробка документації. Розробка логічної моделі даних.	25.10.2020	18.11.2020
Робочий проект	Виготовлення компонентів ПЗ. Розробка фізичної моделі даних.	19.11.2020	10.12.2020
Введення в дію	Дослідне функціонування ПЗ з метою перевірки дієздатності, корегування документації	11.12.2020	14.12.2020

5 Порядок прийому та контролю

Для контролю і прийому повинне бути надана інструкція користувача та опис програми.

Перевірка документації програми здійснюється передставником замовника з метою зафіксувати факт відповідності (або невідповідності) створеного програмного забезпеченню всім пунктам технічної документації.

Порядок контролю і прийому даної розробки здійснюється передставником замовника у присутності передставника розробника згідно з програмою та методикою випробувань шляхом зіставлення характеристик системи з вимогами контракту.

За результатами прийому складається акт, який підписується передставником замовника та передставником розробника та затверджується керівниками організації -замовника і організації -розробника.

Якщо програма не пройшла випробування, складається акт про виявлені помилки та недоліки, що підписується передставниками замовника та розробника, та протокол помилок, до якого вносять виявлені помилки. Виконавець зобов'язаний виправити помилки та недоліки у рядків, не більш ніж 1 місяць з дня випробування та повідомити замовника про повторне проведенні перевірки не пізніше ніж 2 тижні до качану прийому програмного продукту. Після чого проводитися додаткове тестування тієї частини програми, де булі виявлені помилки.

Після виправлення помилок або у разі відсутності таких і при відповідності програми встановленим вимогам підписується акт прийому програмного забезпечення.

ДОДАТОК Б - КОД ПРОГРАМИ

1) Модуль запуску тестів:

```
&НаКлиенте
Перем КонтекстЯдра;
&НаКлиенте
Перем Ожидаем;

&НаКлиенте
Перем ЭтоЗначениеЗаполняетсяПередЗапускомТеста;
&НаКлиенте
Перем ЭтоЗначениеЗаполняетсяПослеЗапускаТеста;
&НаКлиенте
Перем ТекстИсключенияПадающегоТеста;

&НаКлиенте
Процедура Инициализация(КонтекстЯдраПараметр) Экспорт
     КонтекстЯдра = КонтекстЯдраПараметр;
     Ожидаем = КонтекстЯдра.Плагин("УтвержденияBDD");
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьНаборТестов(НаборТестов) Экспорт
     НаборТестов.НачатьГруппу("Выполнение тестов");
     НаборТестов.Добавить("ТестДолжен_ПроверитьВызов_ПередЗапускомТеста
");
     НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТестированияУсп
ешныйМетода");
     НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТестированияПад
ающегоМетода");
     НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТестированияОтс
утствующегоМетода");
     НаборТестов.Добавить("ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста"
);
     НаборТестов.Добавить("ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста_
УПадающегоТеста");
     НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТеста_Когда_Пер
едЗапускаТеста_СОшибкой");
     НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТеста_Когда_Пос
леЗапускаТеста_СОшибкой");

     НаборТестов.НачатьГруппу("Фильтрация");
     НаборТестов.Добавить("ТестДолжен_ПроверитьВыборочныйЗапускТестов_Ф
ильтрЭлементов");
     НаборТестов.Добавить("ТестДолжен_ПроверитьВыборочныйЗапускТестов_Ф
ильтрКонтейнеров");
```

```
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыборочныйЗапускТестов_С
мешанныйФильтр");

	НаборТестов.НачатьГруппу("Статистика");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВРезультатеТестирования_
ЗаполнениеСтатистики");

	НаборТестов.НачатьГруппу("Тесты с параметрами");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыполнениеТеста_ОдинПара
метр");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыполнениеТеста_Нескольк
оПараметров");

	НаборТестов.НачатьГруппу("Режимы выполнения тестов (случайный и
строгий)");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПоУмолчанию_ТестыВып
олняютсяВСлучайномПорядке");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоТестыВыполняютсяСтро
гоПоПорядку");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВК
онтейнереСоСтрогимПорядком_ПриводитКПропускуОставшихсяШагов");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВК
онтейнереСоСтрогимПорядком_ПриводитКПропускуДочернихКонтейнеров");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПадениеТестовВКонтей
нереСоСлучайнымПорядком_НеВлияетНаДругиеТестовыеМетоды");

	НаборТестов.НачатьГруппу("Передача контекста");
	НаборТестов.Добавить("ТестДолжен_ПроверитьПередачуКонтекстаВЗависи
мыхТестах");
	НаборТестов.Добавить("ТестДолжен_ПроверитьЧтоСохранятьКонтекстМожн
оТолькоВРамкахКонтейнераСоСтрогимПорядкомВыполнения");
	НаборТестов.Добавить("ТестДолжен_ПроверитьЧтоПолучатьКонтекстМожно
ТолькоВРамкахКонтейнераСоСтрогимПорядкомВыполнения");
КонецПроцедуры

&НаКлиенте
Процедура ПередЗапускомТеста() Экспорт
	ЭтоЗначениеЗаполняетсяПередЗапускомТеста = Истина;
	ЭтоЗначениеЗаполняетсяПослеЗапускаТеста = Неопределено;
КонецПроцедуры

&НаКлиенте
Процедура ПослеЗапускаТеста() Экспорт
	ЭтоЗначениеЗаполняетсяПослеЗапускаТеста = Истина;
КонецПроцедуры

&НаКлиенте
Процедура ПередЗапускомТеста_СОшибкой() Экспорт
	ВызватьИсключение "ПередЗапускомТеста_СОшибкой";
```

```
КонецПроцедуры

&НаКлиенте
Процедура ПослеЗапускаТеста_СОшибкой() Экспорт
     ВызватьИсключение "ПослеЗапускаТеста_СОшибкой";
КонецПроцедуры


// { Выполнение тестов
&НаКлиенте
Процедура ТестДолжен_ПроверитьВызов_ПередЗапускомТеста() Экспорт
     Ожидаем.Что(ЭтоЗначениеЗаполняетсяПередЗапускомТеста).ЭтоИстина();
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_ПроверитьРезультатТестированияУспешныйМетода()
Экспорт
     ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "УспешныйМетод");
     РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеУспешногоТеста);

     Ожидаем.Что(РезультатТестирования,
"РезультатТестирования").ИмеетТип("Структура");
     Ожидаем.Что(РезультатТестирования.Путь,
"РезультатТестирования.Путь").Равно(ДанныеУспешногоТеста.Путь);
     Ожидаем.Что(РезультатТестирования.ИмяМетода,
"РезультатТестирования.ИмяМетода").Равно(ДанныеУспешногоТеста.ИмяМетода
);
     Ожидаем.Что(РезультатТестирования.Состояние,
"РезультатТестирования.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТ
естов.Пройден);
     Ожидаем.Что(РезультатТестирования.Сообщение,
"РезультатТестирования.Сообщение").Равно("");
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_ПроверитьРезультатТестированияПадающегоМетода()
Экспорт
     ТекстИсключенияПадающегоТеста = "КАБУМ!!!";
     ДанныеПадающегоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "МетодПадающийПоУтверждению");
     РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеПадающегоТеста);
```

```
	Ожидаем.Что(РезультатТестирования,
"РезультатТестирования").ИмеетТип("Структура");
	Ожидаем.Что(РезультатТестирования.Путь,
"РезультатТестирования.Путь").Равно(ДанныеПадающегоТеста.Путь);
	Ожидаем.Что(РезультатТестирования.ИмяМетода,
"РезультатТестирования.ИмяМетода").Равно(ДанныеПадающегоТеста.ИмяМетода
);
	Ожидаем.Что(РезультатТестирования.Состояние,
"РезультатТестирования.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТ
естов.Сломан);
	Ожидаем.Что(РезультатТестирования.Сообщение,
"РезультатТестирования.Сообщение").Содержит(ТекстИсключенияПадающегоТес
та);
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_ПроверитьРезультатТестированияОтсутствующегоМетода() Экспорт
	ДанныеОтсутствующегоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "ОтсутствующийМетод");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеОтсутствующегоТеста);

	Ожидаем.Что(РезультатТестирования,
"РезультатТестирования").ИмеетТип("Структура");
	Ожидаем.Что(РезультатТестирования.Путь,
"РезультатТестирования.Путь").Равно(ДанныеОтсутствующегоТеста.Путь);
	Ожидаем.Что(РезультатТестирования.ИмяМетода,
"РезультатТестирования.ИмяМетода").Равно(ДанныеОтсутствующегоТеста.ИмяМ
етода);
	Ожидаем.Что(РезультатТестирования.Состояние,
"РезультатТестирования.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТ
естов.НеРеализован);
	Ожидаем.Что(РезультатТестирования.Сообщение,
"РезультатТестирования.Сообщение").Содержит(ДанныеОтсутствующегоТеста.И
мяМетода);
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста() Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "УспешныйМетод");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеУспешногоТеста);
```

```
	Ожидаем.Что(ЭтоЗначениеЗаполняетсяПослеЗапускаТеста).ЭтоИстина();
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста_УПадающегоТеста()
Экспорт
	ДанныеПадающегоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "МетодПадающийПоУтверждению");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеПадающегоТеста);

	Ожидаем.Что(РезультатТестирования.Состояние).Равно(КонтекстЯдра.Об
ъект.СостоянияТестов.Сломан);
	Ожидаем.Что(ЭтоЗначениеЗаполняетсяПослеЗапускаТеста).ЭтоИстина();
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_ПроверитьРезультатТеста_Когда_ПередЗапускаТеста_СОшибкой()
Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "УспешныйМетод");
	ДанныеУспешногоТеста.ПередЗапускомТеста =
"ПередЗапускомТеста_СОшибкой";

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеУспешногоТеста);

	Ожидаем.Что(РезультатТестирования.Состояние).Равно(КонтекстЯдра.Об
ъект.СостоянияТестов.НеизвестнаяОшибка);
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_ПроверитьРезультатТеста_Когда_ПослеЗапускаТеста_СОшибкой()
Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ПолучитьИ
спользуемоеИмяФайла(), "УспешныйМетод");
	ДанныеУспешногоТеста.ПослеЗапускаТеста =
"ПослеЗапускаТеста_СОшибкой";

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетодНаКлиенте(ЭтаФорма,
ДанныеУспешногоТеста);
```

```
	Ожидаем.Что(РезультатТестирования.Состояние).Равно(КонтекстЯдра.Об
ъект.СостоянияТестов.НеизвестнаяОшибка);
КонецПроцедуры
// } Выполнение тестов

// { Фильтрация
&НаКлиенте
Процедура ТестДолжен_ПроверитьВыборочныйЗапускТестов_ФильтрЭлементов()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	Фильтр = Новый Массив;
	Фильтр.Добавить(Элемент2.Ключ);
	Фильтр.Добавить(Элемент3.Ключ);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Контейн
ер, Фильтр), "Общее количество тестовых
методов").Равно(Фильтр.Количество());

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер, Фильтр);

	Ожидаем.Что(РезультатТестирования.Состояние,
"Контейнер").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки.Количество(), "Количество
тестовых методов").Равно(Фильтр.Количество());

	РезультатТеста_Элемент1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Элемент1.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент1,
"РезультатТеста_Элемент1").ЭтоНеопределено();
```

```
	РезультатТеста_Элемент2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Ключ,
"Тест2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Состояние, "Тест2
Пройден").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);

	РезультатТеста_Элемент3 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Ключ,
"Тест3.Ключ").Равно(Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Состояние, "Тест3
Пройден").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_ПроверитьВыборочныйЗапускТестов_ФильтрКонтейнеров() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	Контейнер1 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер1");
	Контейнер1.Строки.Добавить(Элемент1);

	Контейнер2 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер2");
	Контейнер2.Строки.Добавить(Элемент2);

	Контейнер3 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер3");
	Контейнер3.Строки.Добавить(Элемент3);

	Корень = ПостроительДереваТестов.СоздатьКонтейнер("Корень");
	Корень.Строки.Добавить(Контейнер1);
	Корень.Строки.Добавить(Контейнер2);
	Корень.Строки.Добавить(Контейнер3);
```

```
	Фильтр = Новый Массив;
	Фильтр.Добавить(Контейнер2.Ключ);
	Фильтр.Добавить(Контейнер3.Ключ);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Корень,
Фильтр), "Общее количество тестовых методов").Равно(2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Корень, Фильтр);

	Ожидаем.Что(РезультатТестирования.Ключ,
"Корень.Ключ").Равно(Корень.Ключ);
	Ожидаем.Что(РезультатТестирования.Состояние,
"Корень.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки.Количество(), "Количество
дочерних узлов").Равно(Фильтр.Количество());

	РезультатТеста_Контейнер1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер1.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер1,
"РезультатТеста_Контейнер1").ЭтоНеопределено();

	РезультатТеста_Контейнер2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Ключ,
"Контейнер2.Ключ").Равно(Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Состояние,
"Контейнер2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройд
ен);
	Ожидаем.Что(РезультатТеста_Контейнер2.Строки[0].Ключ,
"Контейнер2.Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Строки[0].Состояние,
"Контейнер2.Элемент2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТес
тов.Пройден);

	РезультатТеста_Контейнер3 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер3.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер3.Ключ,
"Контейнер3.Ключ").Равно(Контейнер3.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер3.Состояние,
"Контейнер2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройд
ен);
	Ожидаем.Что(РезультатТеста_Контейнер3.Строки[0].Ключ,
"Контейнер3.Элемент3.Ключ").Равно(Элемент3.Ключ);
```

```
	Ожидаем.Что(РезультатТеста_Контейнер3.Строки[0].Состояние,
"Контейнер3.Элемент3.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТес
тов.Пройден);
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_ПроверитьВыборочныйЗапускТестов_СмешанныйФильтр()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	Контейнер1 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер1");
	Контейнер1.Строки.Добавить(Элемент1);
	Контейнер1.Строки.Добавить(Элемент2);

	Контейнер2 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер2");
	Контейнер2.Строки.Добавить(Элемент3);

	Корень = ПостроительДереваТестов.СоздатьКонтейнер("Корень");
	Корень.Строки.Добавить(Контейнер1);
	Корень.Строки.Добавить(Контейнер2);

	Фильтр = Новый Массив;
	Фильтр.Добавить(Элемент2.Ключ);
	Фильтр.Добавить(Контейнер2.Ключ);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Корень,
Фильтр), "Общее количество тестовых методов").Равно(2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Корень, Фильтр);

	Ожидаем.Что(РезультатТестирования.Ключ,
"Корень.Ключ").Равно(Корень.Ключ);
	Ожидаем.Что(РезультатТестирования.Состояние,
"Корень.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки.Количество(), "Количество
дочерних узлов").Равно(2);
```

```
	РезультатТеста_Контейнер1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер1.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер1.Ключ,
"Контейнер1.Ключ").Равно(Контейнер1.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер1.Состояние,
"Контейнер1.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройд
ен);
	Ожидаем.Что(РезультатТеста_Контейнер1.Строки.Количество(),
"Контейнер1 количество дочерних узлов").Равно(1);
	РезультатТеста_Элемент1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТеста_
Контейнер1, Элемент1.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент1,
"РезультатТеста_Элемент1").ЭтоНеопределено();
	РезультатТеста_Элемент2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТеста_
Контейнер1, Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Ключ,
"Контейнер1.Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Состояние,
"Контейнер1.Элемент2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТес
тов.Пройден);

	РезультатТеста_Контейнер2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Ключ,
"Контейнер2.Ключ").Равно(Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Состояние,
"Контейнер2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройд
ен);
	РезультатТеста_Элемент3 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТеста_
Контейнер2, Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Ключ,
"Контейнер2.Элемент3.Ключ").Равно(Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Состояние,
"Контейнер2.Элемент3.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТес
тов.Пройден);
КонецПроцедуры
// } Фильтрация

// { Статистика
&НаКлиенте
Процедура
ТестДолжен_ПроверитьВРезультатеТестирования_ЗаполнениеСтатистики()
Экспорт
```

```
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	УспешныйЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	ПадающийЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"МетодПадающийПоУтверждению");
	ОтсутствующийЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ОтсутствующийМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(УспешныйЭлемент);
	Контейнер.Строки.Добавить(ПадающийЭлемент);
	Контейнер.Строки.Добавить(ОтсутствующийЭлемент);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Контейн
ер), "Общее количество тестовых
методов").Равно(Контейнер.Строки.Количество());

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
	Ожидаем.Что(РезультатТестирования.КоличествоСломанныхТестов).Равно
(1);
	Ожидаем.Что(РезультатТестирования.КоличествоНеРеализованныхТестов)
.Равно(1);
	Ожидаем.Что(РезультатТестирования.ВремяВыполнения).Существует();
КонецПроцедуры
// } Статистика

// { Тесты с параметрами
&НаКлиенте
Процедура ТестДолжен_ПроверитьВыполнениеТеста_ОдинПараметр() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");

	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ТестСложенияСОднимПараметром");
	Элемент1.Параметры.Добавить(Новый Структура("Значение1, Значение2,
Результат", 1, 5, 6));

	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ТестСложенияСОднимПараметром");
```

```
	Элемент2.Параметры.Добавить(Новый Структура("Значение1, Значение2,
Результат", "Раз", "-Два", "Раз-Два"));

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Состояние, "Состояние всех
результатов
тестирования").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_ПроверитьВыполнениеТеста_НесколькоПараметров()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");

	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ТестСложенияСНесколькимиПараметрами");
	Элемент1.Параметры.Добавить(1);
	Элемент1.Параметры.Добавить(5);
	Элемент1.Параметры.Добавить(6);

	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ТестСложенияСНесколькимиПараметрами");
	Элемент2.Параметры.Добавить("Раз");
	Элемент2.Параметры.Добавить("-Два");
	Элемент2.Параметры.Добавить("Раз-Два");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Состояние, "Состояние всех
результатов
тестирования").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
```

```
КонецПроцедуры
// } Тесты с параметрами

// { Режимы выполнения тестов (случайный и строгий)
&НаКлиенте
Процедура
ТестДолжен_Проверить_ЧтоПоУмолчанию_ТестыВыполняютсяВСлучайномПорядке()
Экспорт
     ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
     Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
     Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
     Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

     Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
     Контейнер.Строки.Добавить(Элемент1);
     Контейнер.Строки.Добавить(Элемент2);
     Контейнер.Строки.Добавить(Элемент3);

     МаксимальноеКоличествоПопыток = 100;
     Для Сч = 1 По МаксимальноеКоличествоПопыток Цикл
          РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);
          Если РезультатТестирования.Строки[0].Ключ <> Элемент1.Ключ
Тогда
               Прервать;
          КонецЕсли;
     КонецЦикла;
     Ожидаем.Что(Сч, "Порядок в результатах тестирования должен
отличаться от порядка в дереве
тестов").Меньше(МаксимальноеКоличествоПопыток);
КонецПроцедуры

&НаКлиенте
Процедура ТестДолжен_Проверить_ЧтоТестыВыполняютсяСтрогоПоПорядку()
Экспорт
     ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
     Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
```

```
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.СлучайныйПорядокВыполнения = Ложь;
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Строки[0].Ключ,
"Элемент1").Равно(Элемент1.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[1].Ключ,
"Элемент2").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[2].Ключ,
"Элемент3").Равно(Элемент3.Ключ);
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВКонтейнереСоСтрогимПорядком_
ПриводитКПропускуОставшихсяШагов() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"МетодПадающийПоУтверждению");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.СлучайныйПорядокВыполнения = Ложь;
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);
```

```
	Ожидаем.Что(РезультатТестирования.Строки).ИмеетДлину(2);
	Ожидаем.Что(РезультатТестирования.Строки[1].Ключ,
"Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[1].Состояние,
"Элемент2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Сломан)
;
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВКонтейнереСоСтрогимПорядком_
ПриводитКПропускуДочернихКонтейнеров() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"МетодПадающийПоУтверждению");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	ДочернийКонтейнер =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	ДочернийКонтейнер.Строки.Добавить(Элемент3);

	Корень = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Корень.СлучайныйПорядокВыполнения = Ложь;
	Корень.Строки.Добавить(Элемент1);
	Корень.Строки.Добавить(Элемент2);
	Корень.Строки.Добавить(ДочернийКонтейнер);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Корень);

	Ожидаем.Что(РезультатТестирования.Строки).ИмеетДлину(2);
	Ожидаем.Что(РезультатТестирования.Строки[1].Ключ,
"Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[1].Состояние,
"Элемент2.Состояние").Равно(КонтекстЯдра.Объект.СостоянияТестов.Сломан)
;
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_Проверить_ЧтоПадениеТестовВКонтейнереСоСлучайнымПорядком_НеВ
лияетНаДругиеТестовыеМетоды() Экспорт
```

```
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"МетодПадающийПоУтверждению");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Строки).ИмеетДлину(3);
КонецПроцедуры
// } Режимы выполнения тестов (случайный и строгий)

// { Передача контекста
&НаКлиенте
Процедура ТестДолжен_ПроверитьПередачуКонтекстаВЗависимыхТестах()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");

	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ТестСохраняющийКонтекст_ПервоеЗначение");
	Элемент1.Параметры.Добавить(5);

	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ПолучитьИспользуемоеИмяФайла(),
"ТестИспользующийКонтекст_СуммируетПервоеЗначениеВтороеЗначениеИПроверя
етРезультат");
	Элемент2.Параметры.Добавить(7);
	Элемент2.Параметры.Добавить(12);

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.СлучайныйПорядокВыполнения = Ложь;
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
```

```
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Состояние, "Состояние всех
результатов
тестирования").Равно(КонтекстЯдра.Объект.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_ПроверитьЧтоСохранятьКонтекстМожноТолькоВРамкахКонтейнераСоС
трогимПорядкомВыполнения() Экспорт
	// Ожидаем, что этот контейнер со случайным порядком выполнения
	ОписаниеОшибки = "";
	Попытка
		КонтекстЯдра.СохранитьКонтекст(Истина);
	Исключение
		ОписаниеОшибки = ОписаниеОшибки();
	КонецПопытки;
	Ожидаем.Что(ОписаниеОшибки).Заполнено().Содержит(КонтекстЯдра.Объе
кт.ВозможныеИсключения.СохранятьКонтекстТолькоВСтрогомРежиме);
КонецПроцедуры

&НаКлиенте
Процедура
ТестДолжен_ПроверитьЧтоПолучатьКонтекстМожноТолькоВРамкахКонтейнераСоСт
рогимПорядкомВыполнения() Экспорт
	// Ожидаем, что этот контейнер со случайным порядком выполнения
	ОписаниеОшибки = "";
	Попытка
		КонтекстЯдра.ПолучитьКонтекст();
	Исключение
		ОписаниеОшибки = ОписаниеОшибки();
	КонецПопытки;
	Ожидаем.Что(ОписаниеОшибки).Заполнено().Содержит(КонтекстЯдра.Объе
кт.ВозможныеИсключения.ПолучатьКонтекстТолькоВСтрогомРежиме);
КонецПроцедуры
// } Передача контекста

// Методы нужные для тестов
&НаКлиенте
Функция ЗагрузчикЗаглушка()
	Возврат ЭтаФорма;
КонецФункции

&НаКлиенте
Функция ПолучитьКонтекстПоПути(КонтекстЯдра, Путь) Экспорт
```

```
	Возврат ЭтаФорма;
КонецФункции

&НаКлиенте
Процедура УспешныйМетод() Экспорт
КонецПроцедуры

&НаКлиенте
Процедура МетодПадающийПоУтверждению() Экспорт
	КонтекстЯдра.ВызватьОшибкуПроверки(ТекстИсключенияПадающегоТеста);
	//ВызватьИсключение ТекстИсключенияПадающегоТеста;
КонецПроцедуры

&НаКлиенте
Процедура ТестСложенияСОднимПараметром(Параметры) Экспорт
	Ожидаем.Что(Параметры.Значение1 +
Параметры.Значение2).Равно(Параметры.Результат);
КонецПроцедуры

&НаКлиенте
Процедура ТестСложенияСНесколькимиПараметрами(Слагаемое1, Слагаемое2,
Результат) Экспорт
	Ожидаем.Что(Слагаемое1 + Слагаемое2).Равно(Результат);
КонецПроцедуры

&НаКлиенте
Процедура ТестСохраняющийКонтекст_ПервоеЗначение(ПервоеЗначение)
Экспорт
	КонтекстЯдра.СохранитьКонтекст(ПервоеЗначение);
КонецПроцедуры

&НаКлиенте
Процедура
ТестИспользующийКонтекст_СуммируетПервоеЗначениеВтороеЗначениеИПроверяе
тРезультат(ВтороеЗначение, Результат) Экспорт
	ПервоеЗначение = КонтекстЯдра.ПолучитьКонтекст();
	Ожидаем.Что(ПервоеЗначение + ВтороеЗначение).Равно(Результат);
КонецПроцедуры

&НаСервере
Функция ПолучитьИспользуемоеИмяФайла()
	ОбъектНаСервере = РеквизитФормыВЗначение("Объект");

	Возврат ОбъектНаСервере.ИспользуемоеИмяФайла;
КонецФункции
```

2) Модуль тестування:

```
Перем КонтекстЯдра;
Перем Ожидаем;
```

```
Перем ЭтоЗначениеЗаполняетсяПередЗапускомТеста;
Перем ЭтоЗначениеЗаполняетсяПослеЗапускаТеста;
Перем ТекстИсключенияПадающегоТеста;

Процедура Инициализация(КонтекстЯдраПараметр) Экспорт
	КонтекстЯдра = КонтекстЯдраПараметр;
	Ожидаем = КонтекстЯдра.Плагин("УтвержденияBDD");
КонецПроцедуры

Процедура ЗаполнитьНаборТестов(НаборТестов) Экспорт
	НаборТестов.НачатьГруппу("Выполнение тестов");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВызов_ПередЗапускомТеста
");
	НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТестированияУсп
ешныйМетода");
	НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТестированияПад
ающегоМетода");
	НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТестированияОтс
утствующегоМетода");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста"
);
	НаборТестов.Добавить("ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста_
УПадающегоТеста");
	НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТеста_Когда_Пер
едЗапускаТеста_СОшибкой");
	НаборТестов.Добавить("ТестДолжен_ПроверитьРезультатТеста_Когда_Пос
леЗапускаТеста_СОшибкой");

	НаборТестов.НачатьГруппу("Фильтрация");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыборочныйЗапускТестов_Ф
ильтрЭлементов");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыборочныйЗапускТестов_Ф
ильтрКонтейнеров");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыборочныйЗапускТестов_С
мешанныйФильтр");

	НаборТестов.НачатьГруппу("Статистика");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВРезультатеТестирования_
ЗаполнениеСтатистики");

	НаборТестов.НачатьГруппу("Тесты с параметрами");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыполнениеТеста_ОдинПара
метр");
	НаборТестов.Добавить("ТестДолжен_ПроверитьВыполнениеТеста_Нескольк
оПараметров");

	НаборТестов.НачатьГруппу("Режимы выполнения тестов (случайный и
строгий)");
```

```
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПоУмолчанию_ТестыВыполняютсяВСлучайномПорядке");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоТестыВыполняютсяСтрогоПоПорядку");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВКонтейнереСоСтрогимПорядком_ПриводитКПропускуОставшихсяШагов");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВКонтейнереСоСтрогимПорядком_ПриводитКПропускуДочернихКонтейнеров");
	НаборТестов.Добавить("ТестДолжен_Проверить_ЧтоПадениеТестовВКонтейнереСоСлучайнымПорядком_НеВлияетНаДругиеТестовыеМетоды");

	НаборТестов.НачатьГруппу("Передача контекста");
	НаборТестов.Добавить("ТестДолжен_ПроверитьПередачуКонтекстаВЗависимыхТестах");
	НаборТестов.Добавить("ТестДолжен_ПроверитьЧтоСохранятьКонтекстМожноТолькоВРамкахКонтейнераСоСтрогимПорядкомВыполнения");
	НаборТестов.Добавить("ТестДолжен_ПроверитьЧтоПолучатьКонтекстМожноТолькоВРамкахКонтейнераСоСтрогимПорядкомВыполнения");
КонецПроцедуры

Процедура ПередЗапускомТеста() Экспорт
	ЭтоЗначениеЗаполняетсяПередЗапускомТеста = Истина;
	ЭтоЗначениеЗаполняетсяПослеЗапускаТеста = Неопределено;
КонецПроцедуры

Процедура ПослеЗапускаТеста() Экспорт
	ЭтоЗначениеЗаполняетсяПослеЗапускаТеста = Истина;
КонецПроцедуры

Процедура ПередЗапускомТеста_СОшибкой() Экспорт
	ВызватьИсключение "ПередЗапускомТеста_СОшибкой";
КонецПроцедуры

Процедура ПослеЗапускаТеста_СОшибкой() Экспорт
	ВызватьИсключение "ПослеЗапускаТеста_СОшибкой";
КонецПроцедуры

// { Выполнение тестов
Процедура ТестДолжен_ПроверитьВызов_ПередЗапускомТеста() Экспорт
	Ожидаем.Что(ЭтоЗначениеЗаполняетсяПередЗапускомТеста).ЭтоИстина();
КонецПроцедуры

Процедура ТестДолжен_ПроверитьРезультатТестированияУспешныйМетода()
Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла, "УспешныйМетод");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект, ДанныеУспешногоТеста);
```

```
	Ожидаем.Что(РезультатТестирования,
"РезультатТестирования").ИмеетТип("Структура");
	Ожидаем.Что(РезультатТестирования.Путь,
"РезультатТестирования.Путь").Равно(ДанныеУспешногоТеста.Путь);
	Ожидаем.Что(РезультатТестирования.ИмяМетода,
"РезультатТестирования.ИмяМетода").Равно(ДанныеУспешногоТеста.ИмяМетода
);
	Ожидаем.Что(РезультатТестирования.Состояние,
"РезультатТестирования.Состояние").Равно(КонтекстЯдра.СостоянияТестов.П
ройден);
	Ожидаем.Что(РезультатТестирования.Сообщение,
"РезультатТестирования.Сообщение").Равно("");
КонецПроцедуры

Процедура ТестДолжен_ПроверитьРезультатТестированияПадающегоМетода()
Экспорт
	ТекстИсключенияПадающегоТеста = "КАБУМ!!!";
	ДанныеПадающегоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъек
т.ИспользуемоеИмяФайла, "МетодПадающийПоУтверждению");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект, ДанныеПадающегоТеста);

	Ожидаем.Что(РезультатТестирования,
"РезультатТестирования").ИмеетТип("Структура");
	Ожидаем.Что(РезультатТестирования.Путь,
"РезультатТестирования.Путь").Равно(ДанныеПадающегоТеста.Путь);
	Ожидаем.Что(РезультатТестирования.ИмяМетода,
"РезультатТестирования.ИмяМетода").Равно(ДанныеПадающегоТеста.ИмяМетода
);
	Ожидаем.Что(РезультатТестирования.Состояние,
"РезультатТестирования.Состояние").Равно(КонтекстЯдра.СостоянияТестов.С
ломан);
	Ожидаем.Что(РезультатТестирования.Сообщение,
"РезультатТестирования.Сообщение").Содержит(ТекстИсключенияПадающегоТес
та);
КонецПроцедуры

Процедура
ТестДолжен_ПроверитьРезультатТестированияОтсутствующегоМетода() Экспорт
	ДанныеОтсутствующегоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъек
т.ИспользуемоеИмяФайла, "ОтсутствующийМетод");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект,
ДанныеОтсутствующегоТеста);
```

```
	Ожидаем.Что(РезультатТестирования,
"РезультатТестирования").ИмеетТип("Структура");
	Ожидаем.Что(РезультатТестирования.Путь,
"РезультатТестирования.Путь").Равно(ДанныеОтсутствующегоТеста.Путь);
	Ожидаем.Что(РезультатТестирования.ИмяМетода,
"РезультатТестирования.ИмяМетода").Равно(ДанныеОтсутствующегоТеста.ИмяМ
етода);
	Ожидаем.Что(РезультатТестирования.Состояние,
"РезультатТестирования.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Н
еРеализован);
	Ожидаем.Что(РезультатТестирования.Сообщение,
"РезультатТестирования.Сообщение").Содержит(ДанныеОтсутствующегоТеста.И
мяМетода);
КонецПроцедуры

Процедура ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста() Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъек
т.ИспользуемоеИмяФайла, "УспешныйМетод");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект, ДанныеУспешногоТеста);

	Ожидаем.Что(ЭтоЗначениеЗаполняетсяПослеЗапускаТеста).ЭтоИстина();
КонецПроцедуры

Процедура ТестДолжен_ПроверитьВызов_ПослеЗапускаТеста_УПадающегоТеста()
Экспорт
	ДанныеПадающегоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъек
т.ИспользуемоеИмяФайла, "МетодПадающийПоУтверждению");
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект, ДанныеПадающегоТеста);

	Ожидаем.Что(РезультатТестирования.Состояние).Равно(КонтекстЯдра.Со
стоянияТестов.Сломан);
	Ожидаем.Что(ЭтоЗначениеЗаполняетсяПослеЗапускаТеста).ЭтоИстина();
КонецПроцедуры

Процедура
ТестДолжен_ПроверитьРезультатТеста_Когда_ПередЗапускаТеста_СОшибкой()
Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъек
т.ИспользуемоеИмяФайла, "УспешныйМетод");
	ДанныеУспешногоТеста.ПередЗапускомТеста =
"ПередЗапускомТеста_СОшибкой";

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект, ДанныеУспешногоТеста);
```

```
	Ожидаем.Что(РезультатТестирования.Состояние).Равно(КонтекстЯдра.Со
стоянияТестов.НеизвестнаяОшибка);
КонецПроцедуры

Процедура
ТестДолжен_ПроверитьРезультатТеста_Когда_ПослеЗапускаТеста_СОшибкой()
Экспорт
	ДанныеУспешногоТеста =
КонтекстЯдра.Плагин("ПостроительДереваТестов").СоздатьЭлемент(ЭтотОбъек
т.ИспользуемоеИмяФайла, "УспешныйМетод");
	ДанныеУспешногоТеста.ПослеЗапускаТеста =
"ПослеЗапускаТеста_СОшибкой";

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТестовыйМетод(ЭтотОбъект, ДанныеУспешногоТеста);

	Ожидаем.Что(РезультатТестирования.Состояние).Равно(КонтекстЯдра.Со
стоянияТестов.НеизвестнаяОшибка);
КонецПроцедуры
// } Выполнение тестов

// { Фильтрация
Процедура ТестДолжен_ПроверитьВыборочныйЗапускТестов_ФильтрЭлементов()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	Фильтр = Новый Массив;
	Фильтр.Добавить(Элемент2.Ключ);
	Фильтр.Добавить(Элемент3.Ключ);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Контейн
ер, Фильтр), "Общее количество тестовых
методов").Равно(Фильтр.Количество());
```

```
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер, Фильтр);

	Ожидаем.Что(РезультатТестирования.Состояние,
"Контейнер").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки.Количество(), "Количество
тестовых методов").Равно(Фильтр.Количество());

	РезультатТеста_Элемент1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Элемент1.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент1,
"РезультатТеста_Элемент1").ЭтоНеопределено();

	РезультатТеста_Элемент2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Ключ,
"Тест2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Состояние, "Тест2
Пройден").Равно(КонтекстЯдра.СостоянияТестов.Пройден);

	РезультатТеста_Элемент3 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Ключ,
"Тест3.Ключ").Равно(Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Состояние, "Тест3
Пройден").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
КонецПроцедуры

Процедура
ТестДолжен_ПроверитьВыборочныйЗапускТестов_ФильтрКонтейнеров() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	Контейнер1 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер1");
	Контейнер1.Строки.Добавить(Элемент1);
```

```
	Контейнер2 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер2");
	Контейнер2.Строки.Добавить(Элемент2);

	Контейнер3 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер3");
	Контейнер3.Строки.Добавить(Элемент3);

	Корень = ПостроительДереваТестов.СоздатьКонтейнер("Корень");
	Корень.Строки.Добавить(Контейнер1);
	Корень.Строки.Добавить(Контейнер2);
	Корень.Строки.Добавить(Контейнер3);

	Фильтр = Новый Массив;
	Фильтр.Добавить(Контейнер2.Ключ);
	Фильтр.Добавить(Контейнер3.Ключ);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Корень,
Фильтр), "Общее количество тестовых методов").Равно(2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Корень, Фильтр);

	Ожидаем.Что(РезультатТестирования.Ключ,
"Корень.Ключ").Равно(Корень.Ключ);
	Ожидаем.Что(РезультатТестирования.Состояние,
"Корень.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки.Количество(), "Количество
дочерних узлов").Равно(Фильтр.Количество());

	РезультатТеста_Контейнер1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер1.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер1,
"РезультатТеста_Контейнер1").ЭтоНеопределено();

	РезультатТеста_Контейнер2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Ключ,
"Контейнер2.Ключ").Равно(Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Состояние,
"Контейнер2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТеста_Контейнер2.Строки[0].Ключ,
"Контейнер2.Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Строки[0].Состояние,
"Контейнер2.Элемент2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Про
йден);
```

```
	РезультатТеста_Контейнер3 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер3.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер3.Ключ,
"Контейнер3.Ключ").Равно(Контейнер3.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер3.Состояние,
"Контейнер2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТеста_Контейнер3.Строки[0].Ключ,
"Контейнер3.Элемент3.Ключ").Равно(Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер3.Строки[0].Состояние,
"Контейнер3.Элемент3.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Про
йден);
КонецПроцедуры

Процедура ТестДолжен_ПроверитьВыборочныйЗапускТестов_СмешанныйФильтр()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	Контейнер1 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер1");
	Контейнер1.Строки.Добавить(Элемент1);
	Контейнер1.Строки.Добавить(Элемент2);

	Контейнер2 =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер2");
	Контейнер2.Строки.Добавить(Элемент3);

	Корень = ПостроительДереваТестов.СоздатьКонтейнер("Корень");
	Корень.Строки.Добавить(Контейнер1);
	Корень.Строки.Добавить(Контейнер2);

	Фильтр = Новый Массив;
	Фильтр.Добавить(Элемент2.Ключ);
	Фильтр.Добавить(Контейнер2.Ключ);

	Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Корень,
Фильтр), "Общее количество тестовых методов").Равно(2);
```

```
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Корень, Фильтр);

	Ожидаем.Что(РезультатТестирования.Ключ,
"Корень.Ключ").Равно(Корень.Ключ);
	Ожидаем.Что(РезультатТестирования.Состояние,
"Корень.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки.Количество(), "Количество
дочерних узлов").Равно(2);

	РезультатТеста_Контейнер1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер1.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер1.Ключ,
"Контейнер1.Ключ").Равно(Контейнер1.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер1.Состояние,
"Контейнер1.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТеста_Контейнер1.Строки.Количество(),
"Контейнер1 количество дочерних узлов").Равно(1);
	РезультатТеста_Элемент1 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТеста_
Контейнер1, Элемент1.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент1,
"РезультатТеста_Элемент1").ЭтоНеопределено();
	РезультатТеста_Элемент2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТеста_
Контейнер1, Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Ключ,
"Контейнер1.Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент2.Состояние,
"Контейнер1.Элемент2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Про
йден);

	РезультатТеста_Контейнер2 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТестир
ования, Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Ключ,
"Контейнер2.Ключ").Равно(Контейнер2.Ключ);
	Ожидаем.Что(РезультатТеста_Контейнер2.Состояние,
"Контейнер2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	РезультатТеста_Элемент3 =
КонтекстЯдра.НайтиРезультатТестированияПоИдентификатору(РезультатТеста_
Контейнер2, Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Ключ,
"Контейнер2.Элемент3.Ключ").Равно(Элемент3.Ключ);
	Ожидаем.Что(РезультатТеста_Элемент3.Состояние,
"Контейнер2.Элемент3.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Про
йден);
КонецПроцедуры
```

```
// } Фильтрация

// { Статистика
Процедура
ТестДолжен_ПроверитьВРезультатеТестирования_ЗаполнениеСтатистики()
Экспорт
     ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
     УспешныйЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
     ПадающийЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"МетодПадающийПоУтверждению");
     ОтсутствующийЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ОтсутствующийМетод");
     ОшибочныйЭлемент =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"МетодПадающийПоНеизвестнойОшибке");

     Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
     Контейнер.Строки.Добавить(УспешныйЭлемент);
     Контейнер.Строки.Добавить(ПадающийЭлемент);
     Контейнер.Строки.Добавить(ОтсутствующийЭлемент);
     Контейнер.Строки.Добавить(ОшибочныйЭлемент);

     Ожидаем.Что(КонтекстЯдра.ПолучитьКоличествоТестовыхМетодов(Контейн
ер), "Общее количество тестовых
методов").Равно(Контейнер.Строки.Количество());

     РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

     Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
     Ожидаем.Что(РезультатТестирования.КоличествоСломанныхТестов).Равно
(1);
     Ожидаем.Что(РезультатТестирования.КоличествоНеРеализованныхТестов)
.Равно(1);
     Ожидаем.Что(РезультатТестирования.КоличествоОшибочныхТестов).Равно
(1);
     Ожидаем.Что(РезультатТестирования.ВремяВыполнения).Существует();
КонецПроцедуры
// } Статистика

// { Тесты с параметрами
Процедура ТестДолжен_ПроверитьВыполнениеТеста_ОдинПараметр() Экспорт
```

```
    ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");

    Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ТестСложенияСОднимПараметром");
    Элемент1.Параметры.Добавить(Новый Структура("Значение1, Значение2,
Результат", 1, 5, 6));

    Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ТестСложенияСОднимПараметром");
    Элемент2.Параметры.Добавить(Новый Структура("Значение1, Значение2,
Результат", "Раз", "-Два", "Раз-Два"));

    Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
    Контейнер.Строки.Добавить(Элемент1);
    Контейнер.Строки.Добавить(Элемент2);

    РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

    Ожидаем.Что(РезультатТестирования.Состояние, "Состояние всех
результатов тестирования").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
    Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
КонецПроцедуры

Процедура ТестДолжен_ПроверитьВыполнениеТеста_НесколькоПараметров()
Экспорт
    ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");

    Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ТестСложенияСНесколькимиПараметрами");
    Элемент1.Параметры.Добавить(1);
    Элемент1.Параметры.Добавить(5);
    Элемент1.Параметры.Добавить(6);

    Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ТестСложенияСНесколькимиПараметрами");
    Элемент2.Параметры.Добавить("Раз");
    Элемент2.Параметры.Добавить("-Два");
    Элемент2.Параметры.Добавить("Раз-Два");

    Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
    Контейнер.Строки.Добавить(Элемент1);
```

```
	Контейнер.Строки.Добавить(Элемент2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Состояние, "Состояние всех
результатов тестирования").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
КонецПроцедуры
// } Тесты с параметрами

// { Режимы выполнения тестов (случайный и строгий)
Процедура
ТестДолжен_Проверить_ЧтоПоУмолчанию_ТестыВыполняютсяВСлучайномПорядке()
Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	МаксимальноеКоличествоПопыток = 100;
	Для Сч = 1 По МаксимальноеКоличествоПопыток Цикл
		РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);
		Если РезультатТестирования.Строки[0].Ключ <> Элемент1.Ключ
Тогда
			Прервать;
		КонецЕсли;
	КонецЦикла;
	Ожидаем.Что(Сч, "Порядок в результатах тестирования должен
отличаться от порядка в дереве
тестов").Меньше(МаксимальноеКоличествоПопыток);
КонецПроцедуры

Процедура ТестДолжен_Проверить_ЧтоТестыВыполняютсяСтрогоПоПорядку()
Экспорт
```

```
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.СлучайныйПорядокВыполнения = Ложь;
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Строки[0].Ключ,
"Элемент1").Равно(Элемент1.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[1].Ключ,
"Элемент2").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[2].Ключ,
"Элемент3").Равно(Элемент3.Ключ);
КонецПроцедуры

Процедура
ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВКонтейнереСоСтрогимПорядком_
ПриводитКПропускуОставшихсяШагов() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"МетодПадающийПоУтверждению");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Контейнер.СлучайныйПорядокВыполнения = Ложь;
	Контейнер.Строки.Добавить(Элемент1);
	Контейнер.Строки.Добавить(Элемент2);
	Контейнер.Строки.Добавить(Элемент3);
```

```
	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Строки).ИмеетДлину(2);
	Ожидаем.Что(РезультатТестирования.Строки[0].Ключ,
"Элемент1.Ключ").Равно(Элемент1.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[0].Состояние,
"Элемент1.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.Строки[1].Ключ,
"Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[1].Состояние,
"Элемент2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Сломан);
КонецПроцедуры

Процедура
ТестДолжен_Проверить_ЧтоПадениеОдногоТестаВКонтейнереСоСтрогимПорядком_
ПриводитКПропускуДочернихКонтейнеров() Экспорт
	ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
	Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
	Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"МетодПадающийПоУтверждению");
	Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

	ДочернийКонтейнер =
ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	ДочернийКонтейнер.Строки.Добавить(Элемент3);

	Корень = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
	Корень.СлучайныйПорядокВыполнения = Ложь;
	Корень.Строки.Добавить(Элемент1);
	Корень.Строки.Добавить(Элемент2);
	Корень.Строки.Добавить(ДочернийКонтейнер);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Корень);

	Ожидаем.Что(РезультатТестирования.Строки).ИмеетДлину(2);
	Ожидаем.Что(РезультатТестирования.Строки[1].Ключ,
"Элемент2.Ключ").Равно(Элемент2.Ключ);
	Ожидаем.Что(РезультатТестирования.Строки[1].Состояние,
"Элемент2.Состояние").Равно(КонтекстЯдра.СостоянияТестов.Сломан);
КонецПроцедуры
```

```
Процедура
ТестДолжен_Проверить_ЧтоПадениеТестовВКонтейнереСоСлучайнымПорядком_НеВ
лияетНаДругиеТестовыеМетоды() Экспорт
     ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");
     Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");
     Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"МетодПадающийПоУтверждению");
     Элемент3 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"УспешныйМетод");

     Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
     Контейнер.Строки.Добавить(Элемент1);
     Контейнер.Строки.Добавить(Элемент2);
     Контейнер.Строки.Добавить(Элемент3);

     РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

     Ожидаем.Что(РезультатТестирования.Строки).ИмеетДлину(3);
КонецПроцедуры
// } Режимы выполнения тестов (случайный и строгий)

// { Передача контекста
Процедура ТестДолжен_ПроверитьПередачуКонтекстаВЗависимыхТестах()
Экспорт
     ПостроительДереваТестов =
КонтекстЯдра.Плагин("ПостроительДереваТестов");

     Элемент1 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ТестСохраняющийКонтекст_ПервоеЗначение");
     Элемент1.Параметры.Добавить(5);

     Элемент2 =
ПостроительДереваТестов.СоздатьЭлемент(ЭтотОбъект.ИспользуемоеИмяФайла,
"ТестИспользующийКонтекст_СуммируетПервоеЗначениеВтороеЗначениеИПроверя
етРезультат");
     Элемент2.Параметры.Добавить(7);
     Элемент2.Параметры.Добавить(12);

     Контейнер = ПостроительДереваТестов.СоздатьКонтейнер("Контейнер");
     Контейнер.СлучайныйПорядокВыполнения = Ложь;
     Контейнер.Строки.Добавить(Элемент1);
```

```
	Контейнер.Строки.Добавить(Элемент2);

	РезультатТестирования =
КонтекстЯдра.ВыполнитьТесты(ЗагрузчикЗаглушка(), Контейнер);

	Ожидаем.Что(РезультатТестирования.Состояние, "Состояние всех
результатов тестирования").Равно(КонтекстЯдра.СостоянияТестов.Пройден);
	Ожидаем.Что(РезультатТестирования.КоличествоТестов).Равно(Контейне
р.Строки.Количество());
КонецПроцедуры

Процедура
ТестДолжен_ПроверитьЧтоСохранятьКонтекстМожноТолькоВРамкахКонтейнераСоС
трогимПорядкомВыполнения() Экспорт
	// Ожидаем, что этот контейнер со случайным порядком выполнения
	Ожидаем.Что(КонтекстЯдра)
		.Метод("СохранитьКонтекст",
КонтекстЯдра.ПараметрыМетода(Истина))

	.ВыбрасываетИсключение(КонтекстЯдра.ВозможныеИсключения.СохранятьК
онтекстТолькоВСтрогомРежиме);
КонецПроцедуры

Процедура
ТестДолжен_ПроверитьЧтоПолучатьКонтекстМожноТолькоВРамкахКонтейнераСоСт
рогимПорядкомВыполнения() Экспорт
	// Ожидаем, что этот контейнер со случайным порядком выполнения
	Ожидаем.Что(КонтекстЯдра)
		.Метод("ПолучитьКонтекст")

	.ВыбрасываетИсключение(КонтекстЯдра.ВозможныеИсключения.ПолучатьКо
нтекстТолькоВСтрогомРежиме);
КонецПроцедуры
// } Передача контекста

// Методы нужные для тестов
Функция ЗагрузчикЗаглушка()
	Возврат ЭтотОбъект;
КонецФункции

Функция ПолучитьКонтекстПоПути(КонтекстЯдра, Путь) Экспорт
	Возврат ЭтотОбъект;
КонецФункции

Процедура УспешныйМетод() Экспорт
КонецПроцедуры

Процедура МетодПадающийПоУтверждению() Экспорт
	КонтекстЯдра.ВызватьОшибкуПроверки(ТекстИсключенияПадающегоТеста);
```

```
КонецПроцедуры

Процедура МетодПадающийПоНеизвестнойОшибке() Экспорт
     ВызватьИсключение "Ошибка!!!";
КонецПроцедуры

Процедура ТестСложенияСОднимПараметром(Параметры) Экспорт
     Ожидаем.Что(Параметры.Значение1 +
Параметры.Значение2).Равно(Параметры.Результат);
КонецПроцедуры

Процедура ТестСложенияСНесколькимиПараметрами(Слагаемое1, Слагаемое2,
Результат) Экспорт
     Ожидаем.Что(Слагаемое1 + Слагаемое2).Равно(Результат);
КонецПроцедуры

Процедура ТестСохраняющийКонтекст_ПервоеЗначение(ПервоеЗначение)
Экспорт
     КонтекстЯдра.СохранитьКонтекст(ПервоеЗначение);
КонецПроцедуры

Процедура
ТестИспользующийКонтекст_СуммируетПервоеЗначениеВтороеЗначениеИПроверяе
тРезультат(ВтороеЗначение, Результат) Экспорт
     ПервоеЗначение = КонтекстЯдра.ПолучитьКонтекст();
     Ожидаем.Что(ПервоеЗначение + ВтороеЗначение).Равно(Результат);
КонецПроцедуры
```

ДОДАТОК В - ОПИС ПРОГРАМИ

Програмне забезпечення для хмарного тестування програмного середовища 1С та розробка програми для її реалізації, що розроблялась у рамках магістерської роботи, спрямоване на використання на підприємствах, які займаються розробкою програмного забезпечення.

Дане програмне забезпечення розроблене з метою спрощення роботи IT - компаній при розробці ПЗ та тестування розроблених програмних модулів для середовища 1С: Підприємство.

Функціональним призначенням програмного забезпечення є автоматизація процесів:

- можливість імпортувати звонішній файл з даними (кодом модулю);
- можливість протестувати файл з даними;
- можливість виконати навантажувальне тестування на апаратну частину відділеного сервера;
- можливість протестувати програмний код на «масштабованість».

Для розробки програмного забезпечення було обрано середовище програмування 1С : Підприємство.

При розробці обробника для платформи 1С значну увагу було приділено користувацькому інтерфейсу. Важливим завданням було створити його якомога простішим у використанні і в тій самий година багатофункціональним і зручним, адже ним користуватимуться дуже часто.

Було проведене повне функціональне тестування, а також навантажувальне тестування. Усі виявлені недоліки ПЗ булі зафіксовані в протоколах і усунуті розробником до моменту впровадження програми в дію. Наступні випробування булі успішними та показали повну готовність програми до використання. _

ДОДАТОК Г - ІНСТРУКЦІЯ КОРИСТУВАЧА

Програмне забезпечення автоматизованого хмарного тестування програмного забезпечення 1С та розробка програми для її реалізації рамках магістерської роботи.

Після запуску програмного середовища 1С : Підприємство на виконання, обираємо пункт меню «Файл - Відкрити» та обираємо файл звонішньої обробки «Тестування.epf». Перед користувачем з' являється початкове вікно обробки. Оберемо файл обробки для тестування у файловій системі для демонстрації роботи обробки :

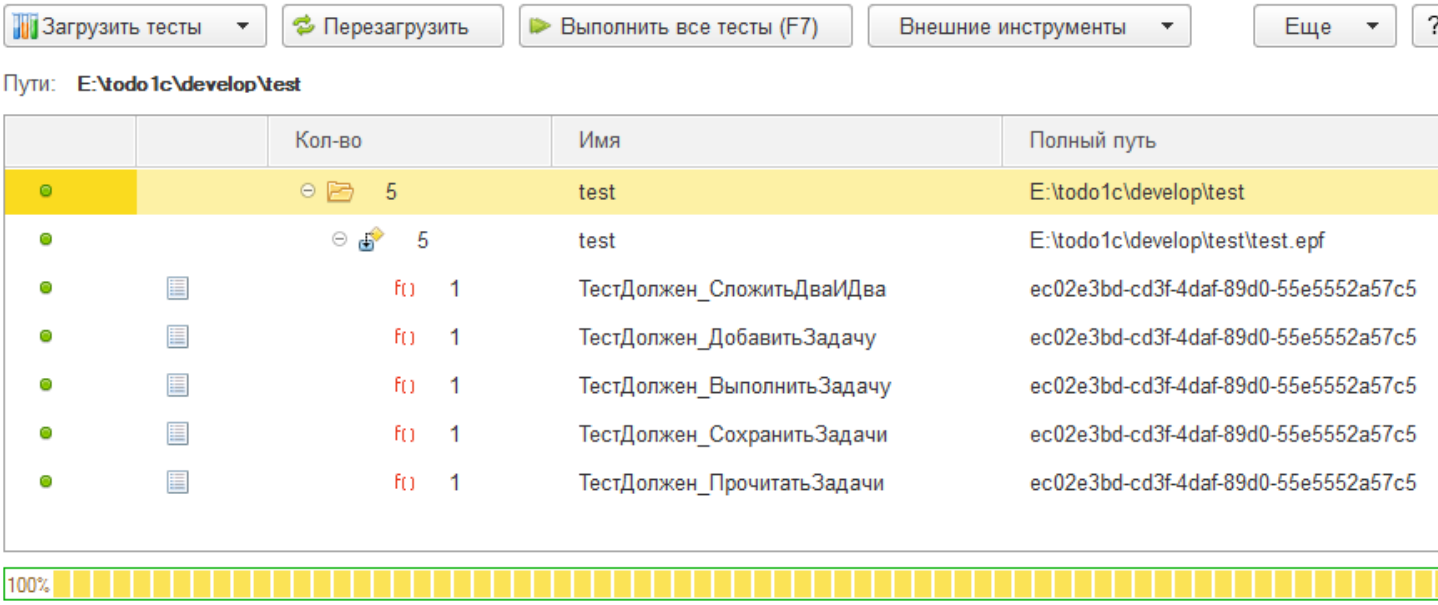


Рисунок Г. 1 - Результат тестування простої обробки

Візьмемо для прикладу більш складний файл обробки.

Вікно з виводом результатів матиме приблизно такий самий вигляд, як і на скріншотах рис.Г. 2 - Г. 4.

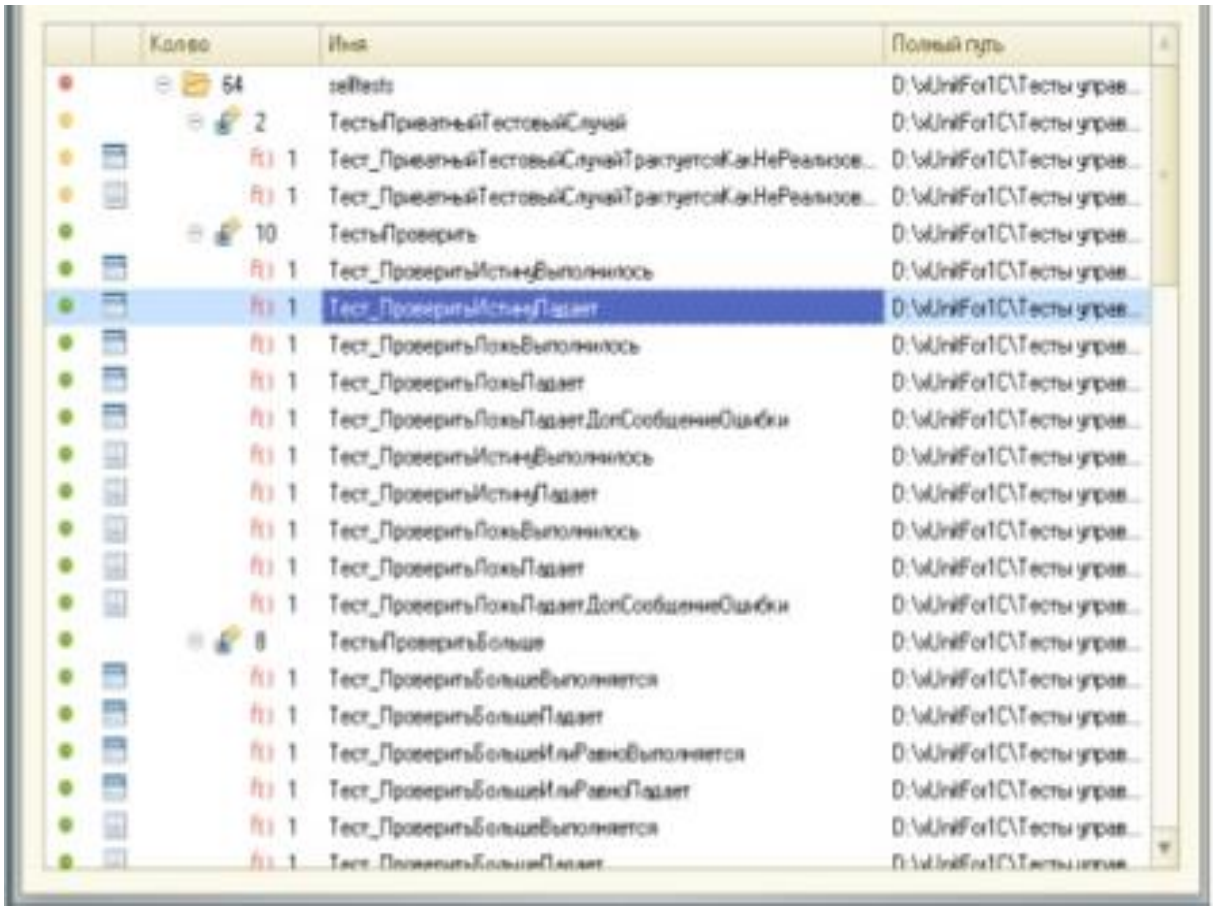


Рисунок Г. 2 - Результат перевірки керованого додатку

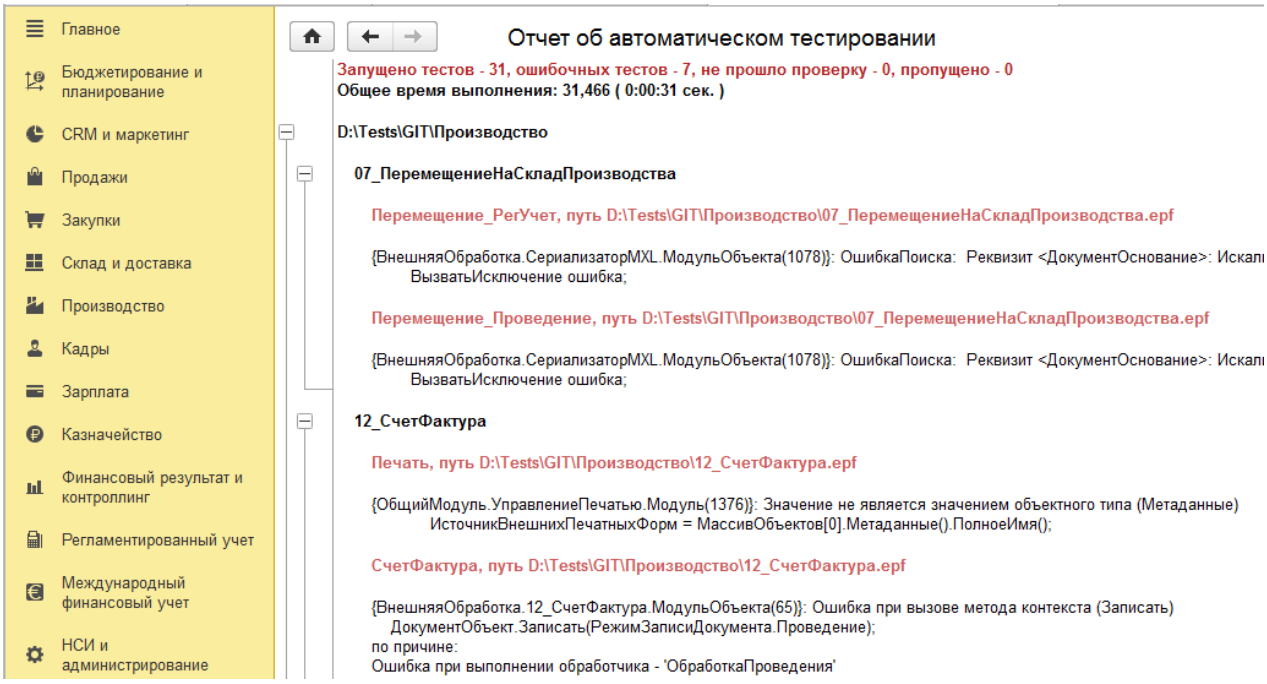


Рисунок Г. 3 - Приклад звіту з перевірки процедур зовнішньої обробки

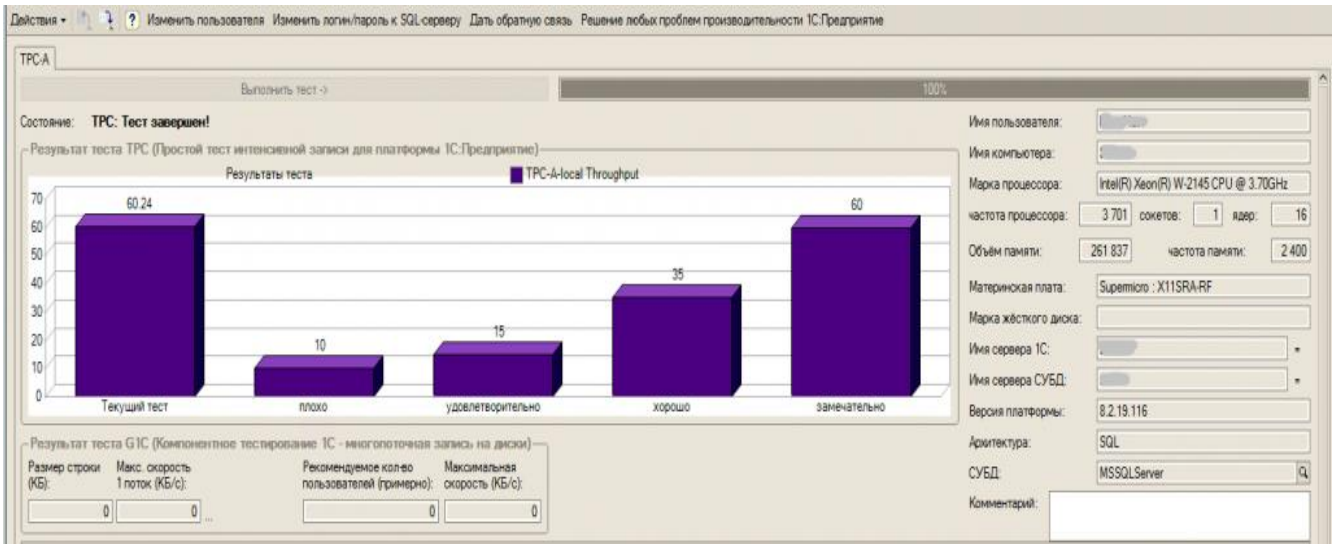


Рисунок Г. 4 - Підключення тесту Гілева до обробки

Програма опрацьовує інформацію поетапно, тому не всі кнопки відразу є активними. Опис функцій та інформаційних блоків, наявних на початковому вікні програми наведене нижче:

1. Кнопка для імпорту файлу з кодом зовнішньої обробки.
2. Кнопка для очищення усіх полів.
3. Кнопка завантаження інших типів тестів.
4. Кнопка повторної перевірки «Перезавантажити».
5. Кнопка запуску виконання всіх тестів по черзі.
6. Кнопка для збереження отриманих результатів «Ще» (у розробці).
7. Кнопка перегляданню інформації при наведенні курсора на полі введенню або кнопку.
8. Блок параметрів вихідних даних (результатів тестування).

Порядок дій для тестування файлів зовнішніх обробок програмного середовища 1С : Підприємство:

1. Натиснути кнопку «Імпорт файлу», відкривається вікно вибору файлу. Обирається необхідний файл.

2. Натиснути кнопку «Запустити усі тести». Чекаємо на результат в нижньому блоці робочої області;

3. Після натискання кнопки «Перезавантажити», програма скидає всі значення. Повертаємося до п.1.

4. Зберігаємо дані у вигляді файлу csv після натискання кнопки «Ще - Зберегти дані».

ДОДАТОК Д - ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1 Об' єкт випробування

Об' єктом випробування є програмне забезпечення для хмарного тестування програмного забезпечення 1С, розроблене у рамках даної роботи.

2 Мета випробування

Метою проведення випробування є перевірка працездатності даного програмного продукту, перевірка відповідності характеристик та вимог розробленої програми, викладених у технічному завдані, приведення прикладу роботи та отримання відповідних навичок.

3 Вимоги до додатку

При проведенні випробувань функціональні характеристики (можливості) програми підлягають перевірці на відповідність вимогам, викладеним у пункті «Вимоги до функціональних характеристик» технічного завдання.

4 Вимоги до програмної документації

4.1 Склад програмної документації, пропонованої на випробування

Програмна документація повинна включати в собі наступні документи:

1) Технічне завдання;

2) Текст програми;

3) Опис програми;

4) Інструкція користувача;

5) Програма та методика випробувань.

4.2 Спеціальні вимоги

Спеціальні вимоги до програмної документації не висуваються.

5 Засоби і порядок випробувань

5.1 Технічні засоби, що використовуються під година випробувань

Склад технічних засобів, що використовувалися під година випробувань :

– CPU: Intel Pentium 4405Y (1.2 ГГц);
– RAM: 1 GB;
– Жорсткий диск 500 ГБ;
– Роздільна здатність екрану - 1920×1080 пікселів;
– Клавіатура 101/102 - х клавішна русявий / лат;
– Маніпулятор «миша».

5.2 Програмні засоби, що використовуються під година випробувань

Програмне забезпечення не перед' являє ніяких вимог до апаратної платформи.

Склад програмних засобів, що використовувалися під година випробувань :

Операційні системи:

- Windows 10 Pro x64.

5.3 Порядок проведення випробувань

Порядок проведення випробувань складається з перевірки вимог до функціональних характеристик програмного продукту та перевірки вимог до програмної документації.

6 Методи випробувань

6.1 Методика проведення перевірки вимог до програмної документації

Перевірка дотримання вимог програмної документації на програмний продукт виконується візуально передставником служби, відповідальної за експлуатацію. У ході перевірки зіставляється склад і комплектність програмної документації,

представленої розробником, з переліком програмної документації, наведеним у пункті «Склад програмної документації, пропонованої на випробування» цього документу.

Перевірка вважається завершеною у випадку відповідності складу та комплектності програмної документації, представленої розробником, переліком програмної документації, наведеному у зазначеному вище пункті.

6.2 Методика проведення перевірки вимог до функціональних характеристик програмного продукту

Перевірка працездатності програми виконується згідно з пунктом «Вимоги до функціональних характеристик» технічного завдання.

Перевірка вважається завершеною у разі відповідності складу і послідовності дій, при виконанні даної перевірки, вказаною вище пункту технічного завдання.

У процесі тестування була перевірена функціональність програмного забезпечення хмарного тестування файлів 1С та розробка програми для реалізації. У таблиці Д.1, що наведена нижче, вказано перелік випробувань основних функціональних можливостей.

Таблиця Д.1 - Методика випробування

№	Дія	Очікуваний результат	Результат перевірки	Зауваження
1	2	3	4	5
1	Випробування на імпорт непідтримуваного формату файлу	Виводиться повідомлення про помилку вхідного файлу	Виконано	
2	Випробування на імпорт коректного формату файлу	У результаті файл успішно завантажився та у вікні було виведено його назву.	Виконано	
3	Випробування на тестування параметрів без завантаження файлу	Вікно з результатами порожнє порожнє. Програма очікує імпорту файлу для тестування	Виконано	

Продовження табл. Д.1

1	2	3	4	5
4	Випробування на тестування порожнього файлу зовнішньої обробки	Помилка зчитування файлу зовнішньої обробки	Виконано	
6	Випробування на експорт до файлу csv без проведення тестування	Повідомлення про відсутність данних для відвантаження	Виконано	
7	Випробування на експорт даних до файлу csv після успішного тестування	Випливає діалогове вікно з пропозицією оберни каталог для збереження файлу з даними csv	Виконано	
8	Випробування на правиьність очищення полів з даними	Вікна та поля очищені. Імпорт наступного файлу для тестування успішно прийнято	Виконано	