Дoклад 03

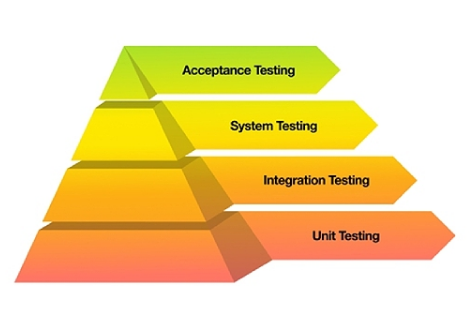
По Софтуерно инженерство

Изготвил: Валентин Кьосев 12 ,,а‘‘ клас

1. Какво е unit test, за какво служи? Дайте примери.
2. Какво е Регресивно тестване, за какво служи? Дайте примери.
3. Какво е Mocking при тестване, за какво служи? Дайте примери.
4. Какво е Интеграционно тестване, за какво служи? Дайте примери.
5. Какво е Софтуерна документация, за какво служи? Дайте примери.
6. **Какво е unit test, за какво служи? Дайте примери.**

Това е процес в програмирането, използван за тестването на отделни компоненти (единици, units) от кода на софтуера. Тези компоненти представляват най-малката част, съществуваща в едно приложение, която позволява да бъде тествана индивидуално. Крайната цел е изолирането на конкретно парче код и установяването дали компонентът работи правилно. Когато става дума за ООП, тези компоненти обикновено са цели класове, а понякога и отделни методи, а в процедурното – най-често са отделни процедури или функции.

Компонентното тестване е ключов момент от разработката на софтуер, тъй като позволява ранното засичане на проблеми и несъвършенства в кода, чието откриване и отстраняване на по-късен етап може да бъде по-сложно и времеемко. Както споменах, това е първото ниво в софтуерното тестване и се изпълнява преди всички останали видове тестове. Всички сценарии се тестват индивидуално и в изолирана среда, за да се премахне възможността за зависимости в кода.

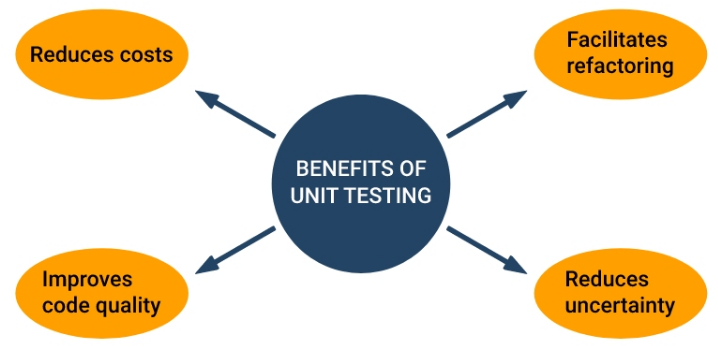


Unit testing процесът е част от т.нар. методология за разработка чрез тестове (test-driven development или TDD), която изисква и създаването на автоматизирани unit тестове. Те дефинират изискванията за кода, преди той да бъде написан. Разбира се, unit testing процесът може да се извършва и ръчно. Но за целите на автоматизираното тестване е нужно подходяща рамка, чрез която да се създават тестове сценарии.

**Unit testing: основни предимства и ползи**

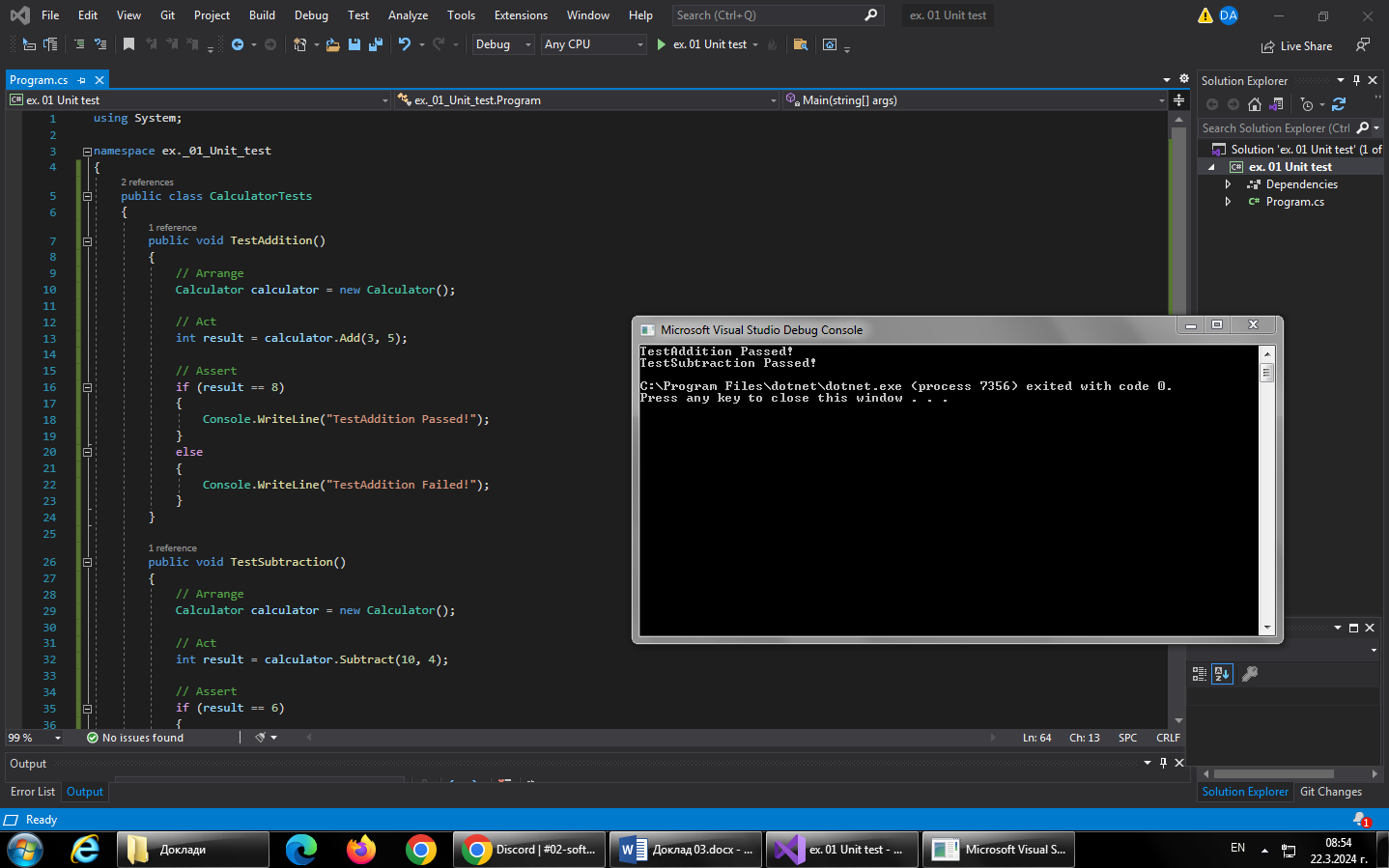
Софтуерното тестване е от изключително значение за качествената разработка, а unit testing процесът е част от работата на всеки QA специалист. Независимо от продукта, по който работиш, и спрямо каква методология, процесът трябва да е част от практиката по няколко причини:

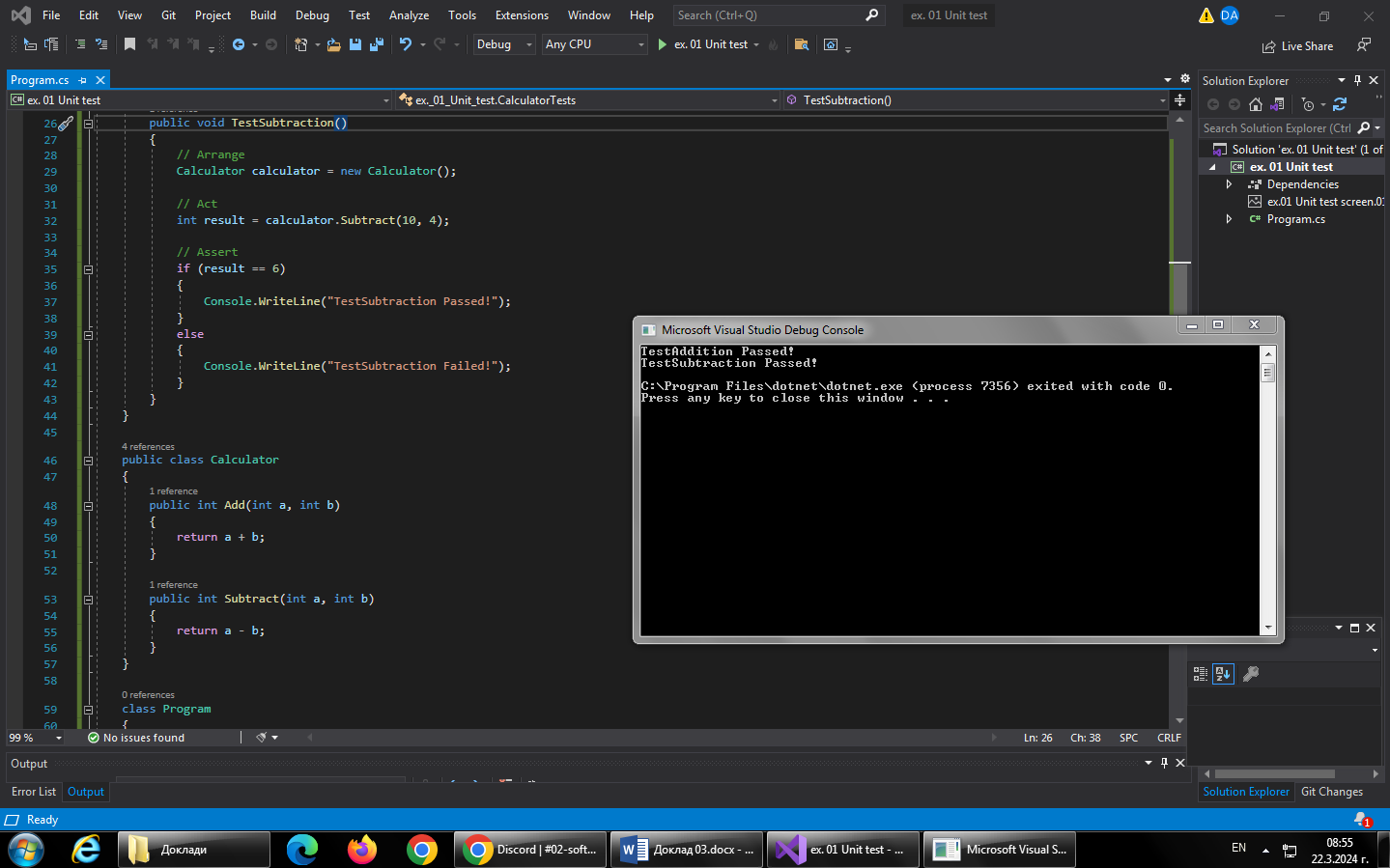
* **Ранно засичане на проблеми** – unit тестовете ти помагат да откриваш дефекти и проблеми в ранните етапи от софтуерната разработка.
* **Намалени разходи** – рано откритите дефекти значително намаляват разходите, които съпътстват откриването и отстраняване на дефекти в по-напреднали етапи.
* **Улеснени промени** – кодът може да бъде рефакториран по-късно, а тестовете създават сигурност, че компонентите ще продължат да работят правилно.
* **Опростена интеграция** – unit тестовете намаляват несигурността по отношение на работата на компонентите. Това улеснява следващото ниво от тестването на софтуера, а именно – integration testing.
* **Преизползваемост на кода** – при unit testing процеса, отделните компоненти биват самостоятелно тествани в изолирана среда. Това означава, че кодът на компонента е надежден и може да бъде преизползван.

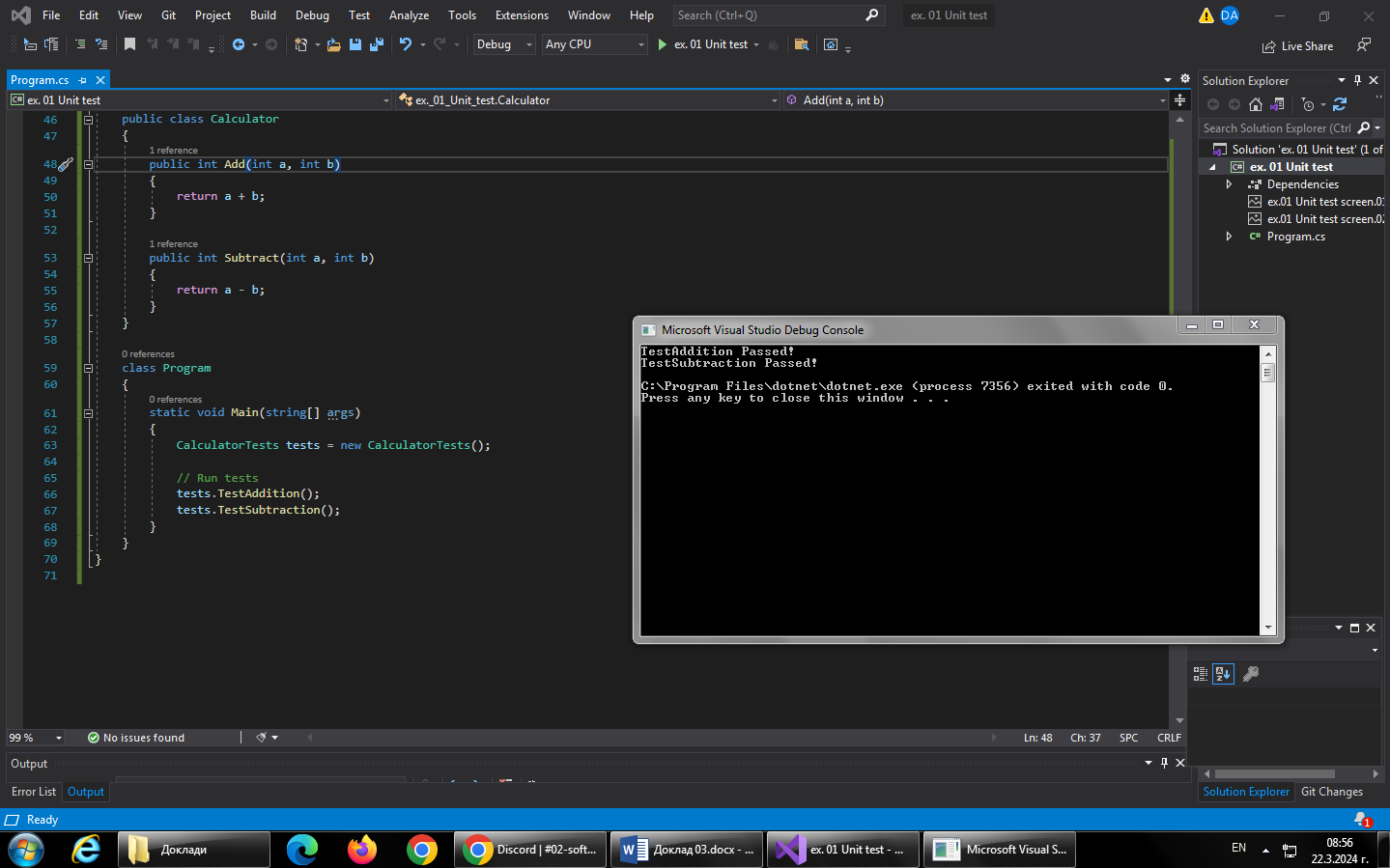


**Пример:**

**Проста имплементация на тестовете в C#:**

****

****

****

Тук имаме клас CalculatorTests, където се намират методите за тестване на функционалността на класа Calculator. След това в Main метода на програмата се създава инстанция на CalculatorTests и се извикват тестовете чрез методите TestAddition и TestSubtraction. Резултатите се извеждат в конзолата.

1. **Какво е Регресивно тестване, за какво служи? Дайте примери.**

Ако софтуерът губи функционалност поради въвеждането на нови или променени функции, се казва, че той е преминал в по-слабо развито състояние. Дори незначителни промени в софтуера или в оригиналния код могат да доведат до значителни грешки, като сривове, проблеми и частична или пълна загуба на функционалност.

**Тестването за регресия** се използва за откриване на тези грешки и за възстановяване на стабилността на приложението. Процесите на функционално и нефункционално тестване оценяват въздействието на новите функции върху съществуващия код.

В много процеси на регресионно тестване се използват данни от тестови сценарии, проведени преди въвеждането на текущия кръг от промени. Например предишни функционални тестове, тестове на блоковете, интеграционни тестове и тестове за проверка на изграждането могат да бъдат интегрирани в регресионните тестове, което позволява проверените резултати от по-ранни етапи на цикъла на разработка да помогнат за диагностицирането на неочаквани текущи проблеми.

Тестването за регресия предоставя важна информация по време на целия цикъл на разработване, включително по време на изграждането на системата, както и при поддръжка след пускането ѝ на пазара. Следните сценарии обикновено изискват регресионно тестване:

* Изпълнение на функциите
* Промени в кодовата база
* По време на закъснения
* След появата на други грешки

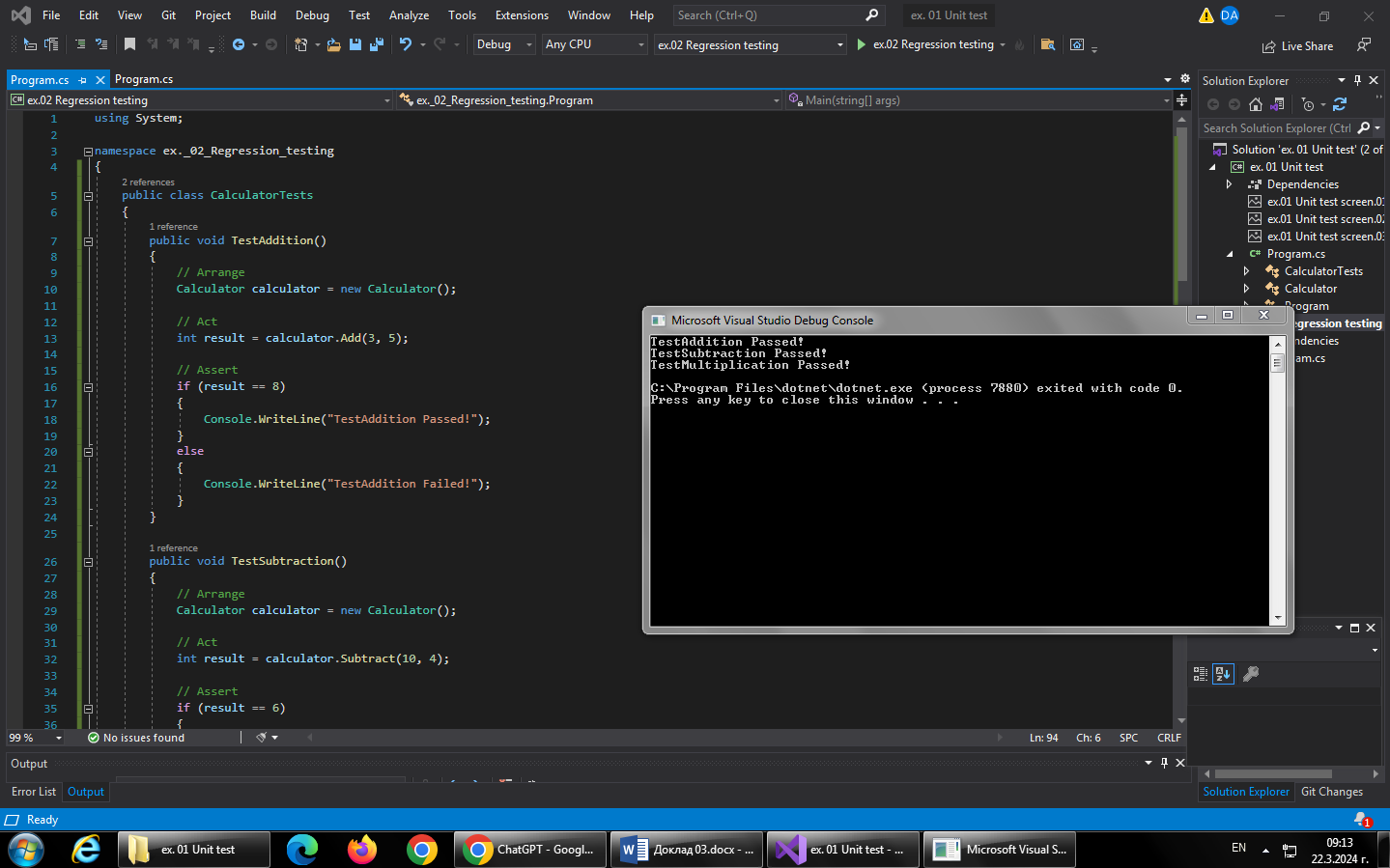
**Предимства на регресионното тестване**

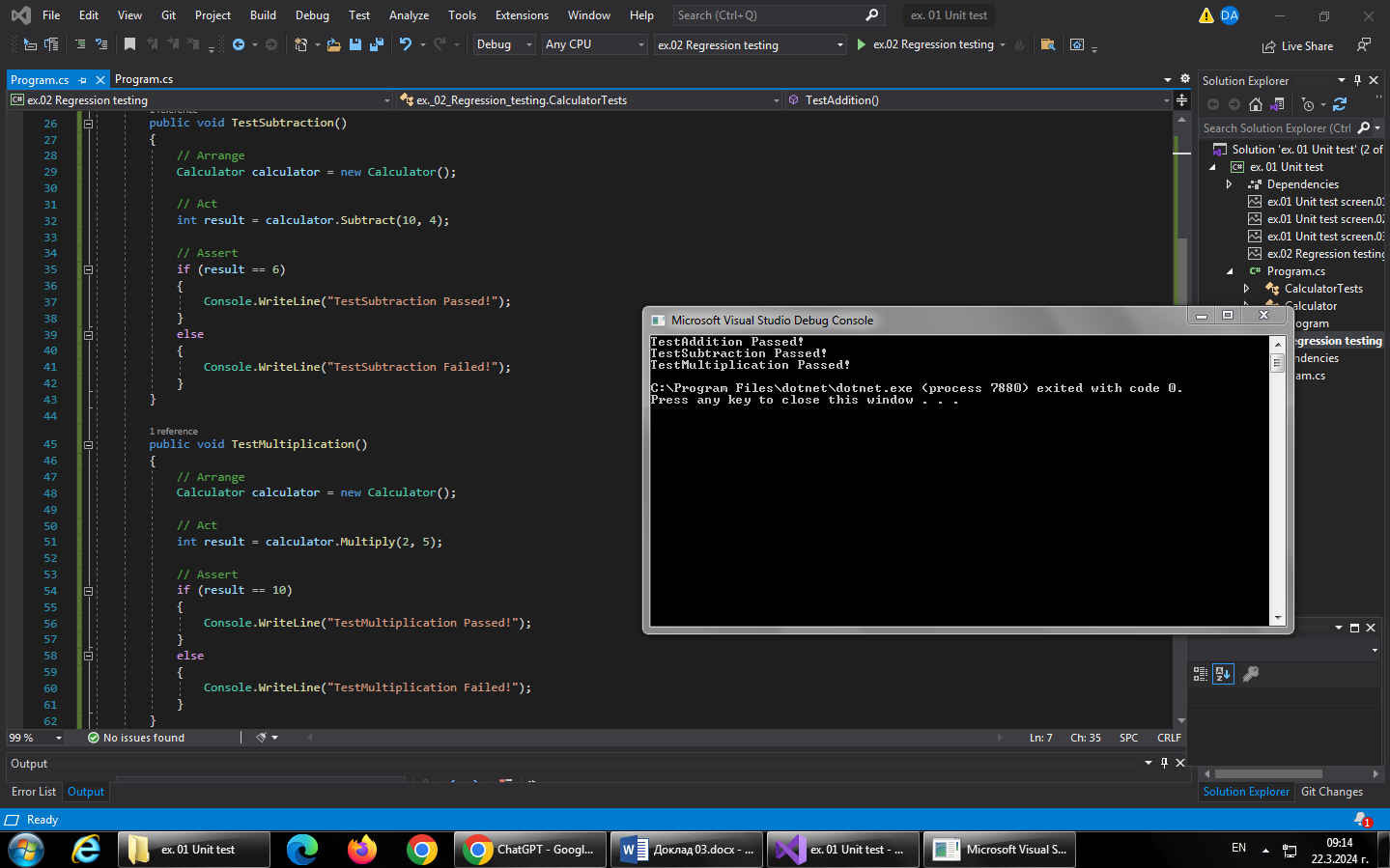
Тестването на регресията е от полза на всеки етап от жизнения цикъл на разработката на софтуер. Очевидното предимство е, че регресионните тестове гарантират безпроблемното функциониране на софтуера след корекция на кода или въвеждане на нова функция. Освен това има и други предимства, които трябва да се вземат предвид.

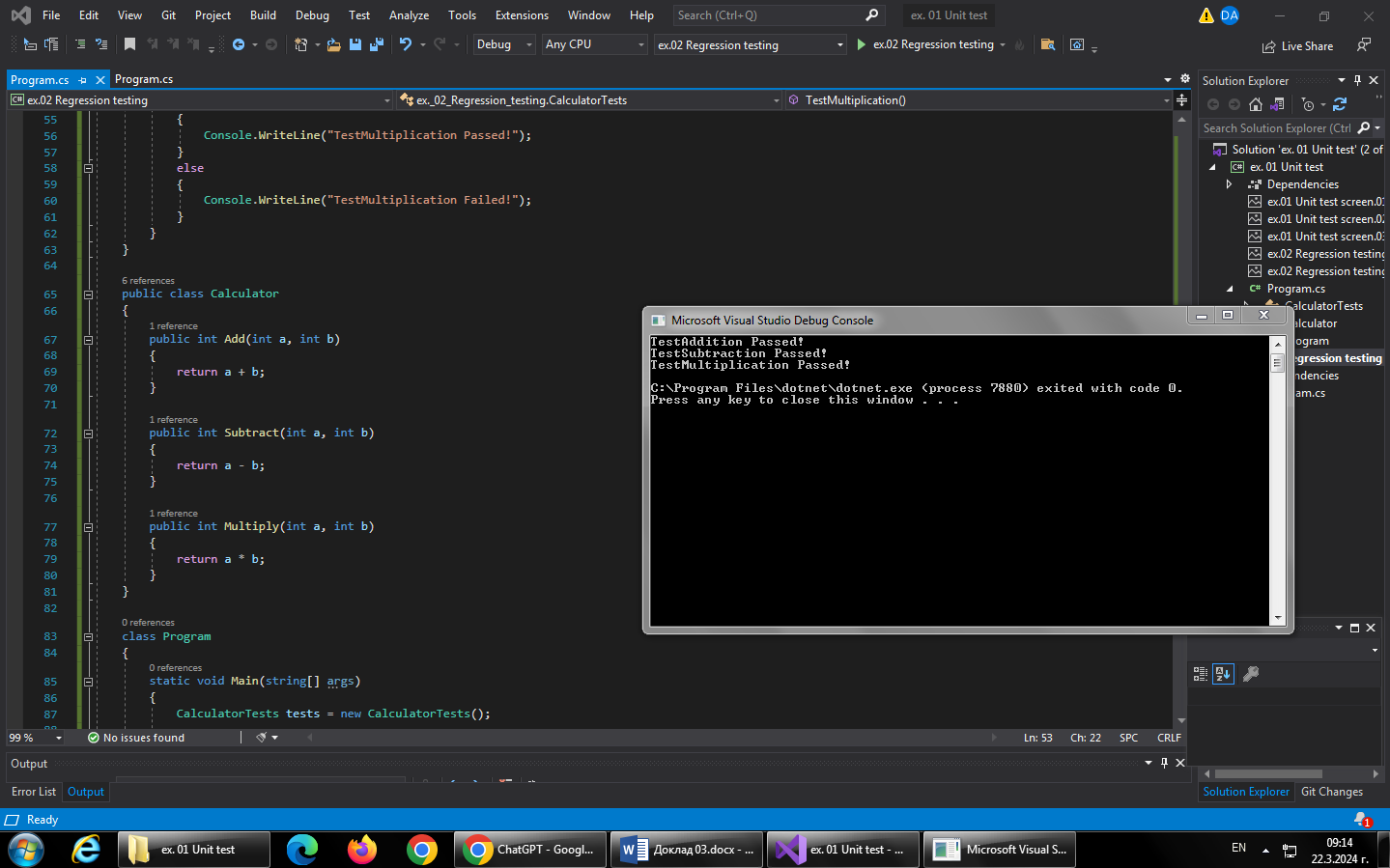
* Незабавно открийте грешки
* Намаляване на ненужните разходи
* Внедряване на непрекъсната интеграция

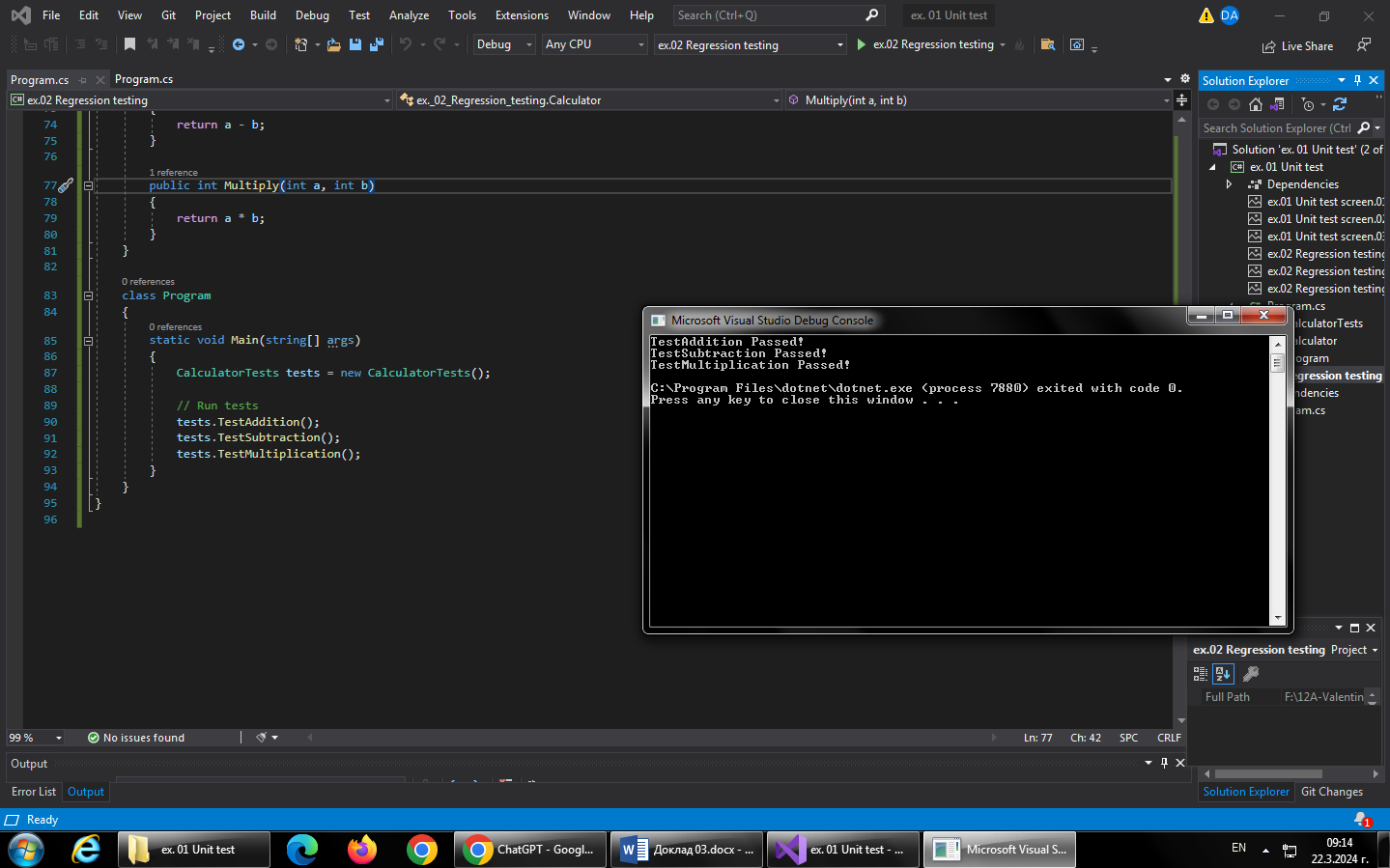
**Пример:**

**пример на регресивно тестване в C# за прост клас Calculator, който има методи за събиране и изваждане:**

****

****

****

****

Тук имаме клас Calculator, който има методи за събиране, изваждане и умножение. Класът CalculatorTests съдържа методи за тестване на тези функционалности.

* TestAddition(): Тест за проверка на метода за събиране.
* TestSubtraction(): Тест за проверка на метода за изваждане.
* TestMultiplication(): Нов добавен тест за проверка на метода за умножение.

В Main метода на програмата се създава инстанция на CalculatorTests и се извикват всички тестове. При стартиране на програмата, ако всички тестове преминат успешно, ще видите съобщения "Passed!" за всяко от тях. Ако има някаква грешка, ще видите "Failed!" съобщение за съответния тест.

1. **Какво е Mocking при тестване, за какво служи? Дайте примери.**

Mocking е техника, използвана за изолиране на тествания модул чрез замяна на зависимости с обекти, които можете да контролирате и проверявате.

Зависимостта може да бъде всичко, от което зависи вашето устройство, но обикновено това е модул, който модулът импортира.

Един тестван модул може да има зависимости от други модули. За да изолирате поведението на единицата, искате да замените другата единица с имитации, които симулират поведението на истинската единица. Това е полезно, ако реалните единици са непрактични за включване в теста на единица.

Накратко, mocking е създаване на единица, която симулира поведението на истинска единица.

**Тестови двойници - dummy, fake, stub, mock, spies**

При тестването има няколко дефиниции на обекти, които не са реални. Общият термин е двоен тест .

Тестов дубльор е обект, който може да замести реален обект в тест, подобно на това как дубльорът замества актьор във филм

Има пет основни вида тестови двойници: dummy, fake, stub, mock, spies

**Stubs**

Stub е обект, който съдържа предварително дефинирани данни и ги използва, за да отговаря на повиквания по време на тестове. Използва се, когато не можем или не искаме да включим обекти, които биха отговорили с реални данни или имат нежелани странични ефекти.

Примерно може да бъде обект, който трябва да вземе някои данни от базата данни, за да отговори на извикване на метод. Вместо реалния обект, въведохме мъниче, което връща твърдо кодираните данни, които трябва да бъдат върнати.

**Mocks**

Mocks са много подобни на Stubs, но основната разлика е, че Mocks са по-„базирани на взаимодействие“. Това означава, че ние не просто очакваме макетът да върне някаква стойност, но да приемем, че се прави определен ред от извиквания на метод.

Mocks са обекти, които регистрират повикванията, които получават.

В тестово твърдение можем да проверим на Mocks, че всички очаквани действия са извършени.

**Fake**

Тези обекти всъщност имат пълна работеща реализация в тях. Но имплементацията, осигурена в тях, е някакъв вид пряк път, който ни помага в нашата задача за тестване на единици, и този пряк път го прави неспособен в производството.

Чудесен пример за това е обектът на базата данни в паметта, който можем да използваме само за нашите тестови цели, докато използваме обекта на реалната база данни в производството.

**Spies**

Шпионите са малки обекти, които също записват някаква информация въз основа на това как са били наречени. Една форма на това може да бъде имейл услуга, която записва колко съобщения е изпратено.

**Dummy**

Когато използваме обект да застане на мястото на реален обект, но никога не използваме обекта, тогава обектът се нарича манекен. Обикновено се прави, за да се попълни списъкът с параметри, така че кодът да се компилира и компилаторът да остане доволен.

1. **Какво е Интеграционно тестване, за какво служи? Дайте примери.**

Интеграционното тестване е съществен аспект от софтуерното тестване, който има за цел да оцени доколко ефективно различните приложения се интегрират заедно.

Повечето съвременни предприятия разчитат на множество различни софтуерни модули всеки ден, а интеграцията позволява на тези приложения да работят заедно, за да подобрят ефективността и да рационализират работните процеси.

Тестването на интеграцията е важно, тъй като безпроблемната интеграция е това, което прави софтуерните модули ефективни. Когато всеки софтуерен модул е програмиран от различен разработчик и използва напълно различна логика на програмиране, няма причина да се смята, че отделните модули ще се интегрират гладко от самото начало.

Тестването на интеграцията позволява на ИТ специалистите да оценят колко добре работят заедно различните модули и да въведат промени за повишаване на ефективността им.

**Какво представлява интеграционното тестване?**

Значението на интеграционното тестване се отнася до процеса на тестване на интерфейсите между два компонента или софтуерни модула, за да се оцени как се прехвърлят данните между тях.

Стратегиите за тестване на интеграцията позволяват на екипите за разработване и ИТ специалистите да откриват дефекти, които могат да се появят при интегрирането на два или повече софтуерни модула, както и да оценяват цялостното съответствие и функциониране на комбинираните софтуерни елементи.

Интеграционното тестване обикновено се извършва след тестването на единици, което включва тестване на отделни модули и единици. След като е установено, че всяко звено работи самостоятелно, интеграционното тестване оценява как работят всички звена, когато се комбинират.

Интеграционното тестване е поетапен процес, който обикновено изисква от тестерите да интегрират модулите един по един и да извършват тестване на всяка стъпка.

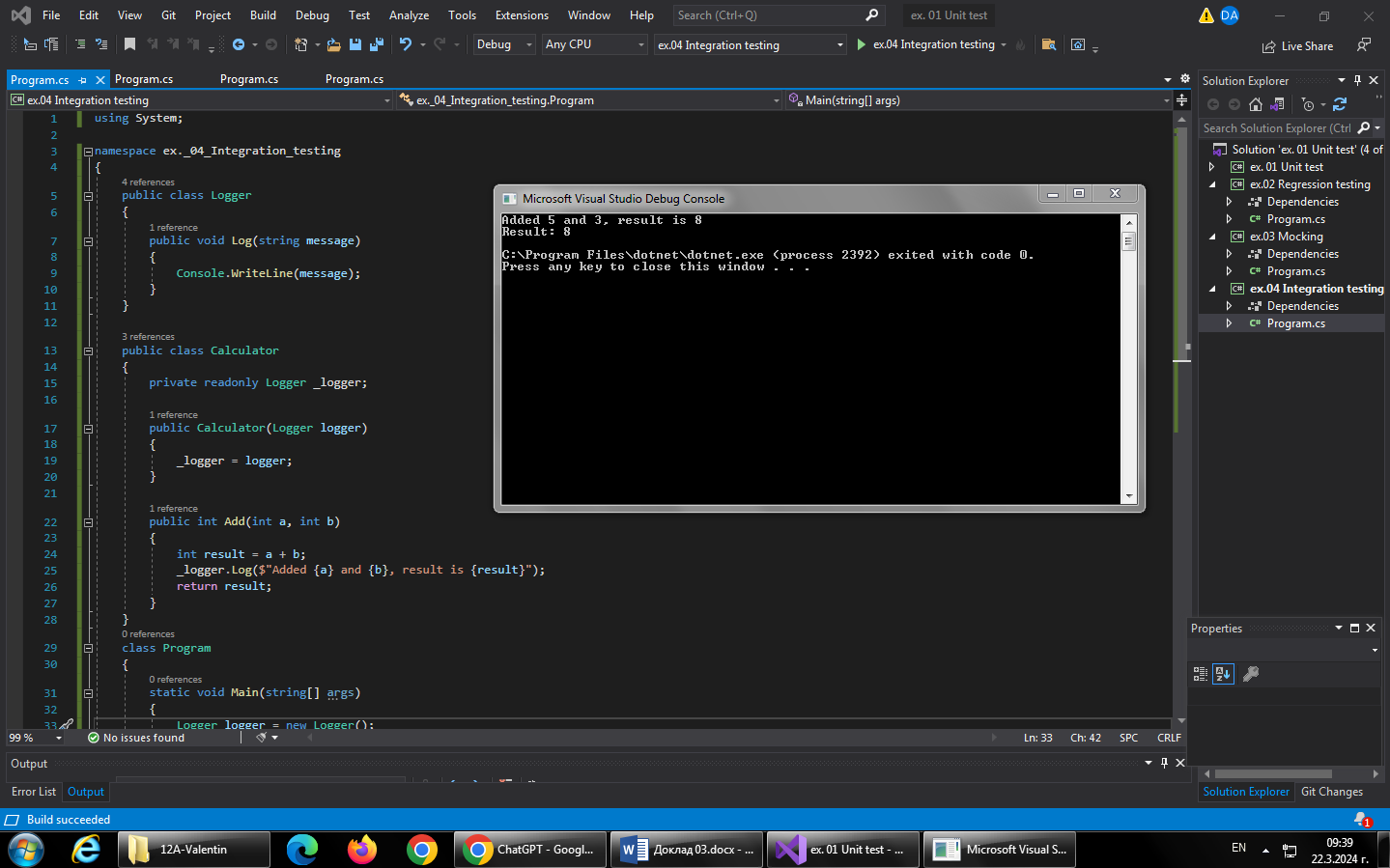
Интеграционните тестове зависят от добре дефинирана спецификация на интерфейса между тестваните компоненти. Тези тестове трябва да бъдат автоматизирани в максимална степен, така че да могат да се изпълняват често, за да се откриват проблемите в началото, преди да се превърнат в сложни проблеми, чието отстраняване отнема време и ресурси на по-късен етап от разработката.

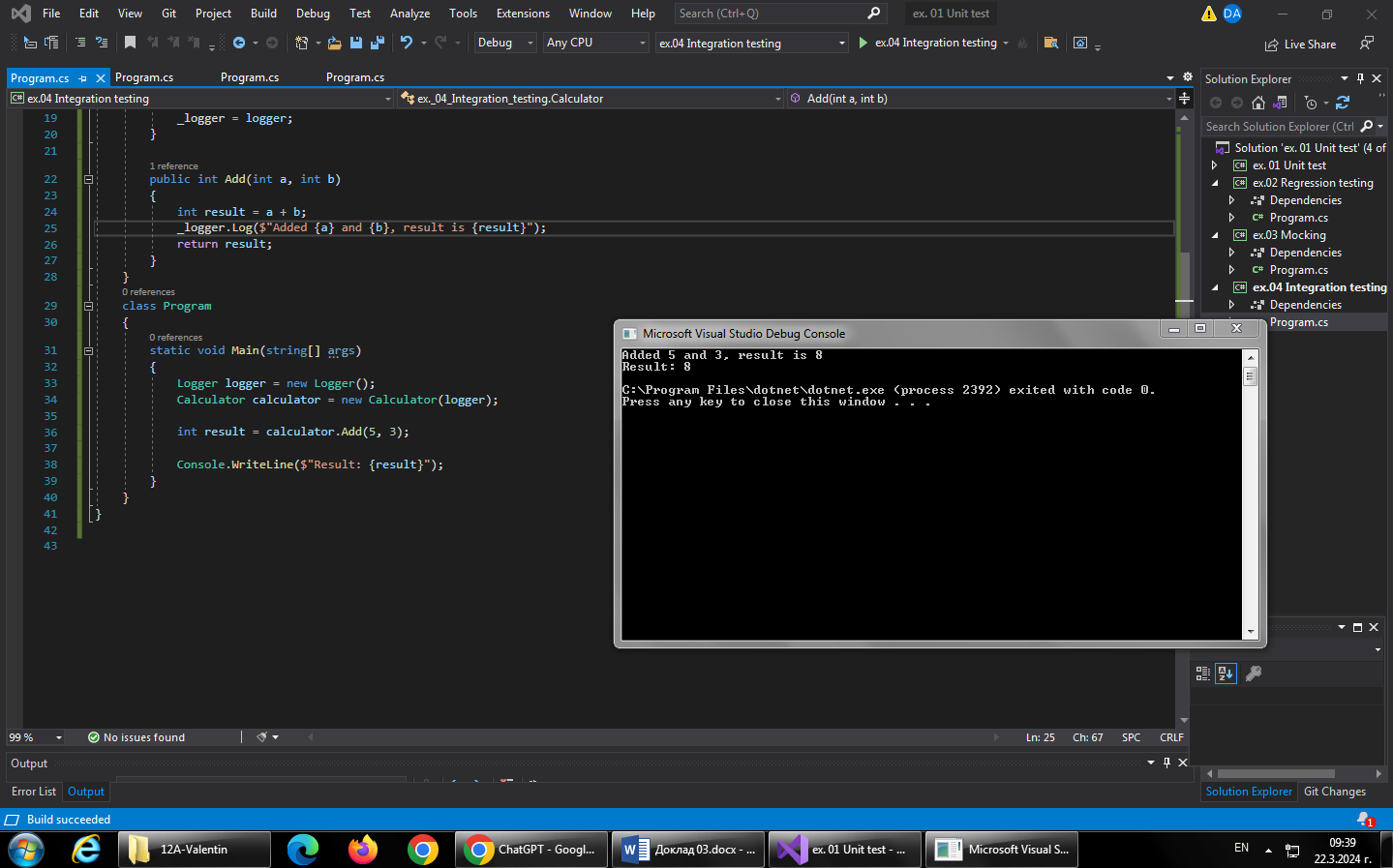
Интеграционното тестване е вид софтуерно тестване, което гарантира, че всички компоненти на приложенията работят заедно, както се очаква.

Целта на интеграционното тестване е да се провери дали интеграцията на различните модули и компоненти в дадено приложение отговаря на изискванията на потребителя, както и на техническите изисквания и изискванията за производителност на организацията.

**Пример:**

**Прост пример на интеграционно тестване без използване на конкретни тестови фреймуърки, можем да напишем прост конзолен пример за два класа, които работят заедно. Имаме клас Calculator, който зависи от клас Logger за логиране на операциите:**

****

****

В този случай Calculator извиква метода Log на Logger при извършването на операцията.

**В реален проект бихте използвали специализиран софтуерен тестови фреймуърк, който би автоматизирал този процес и предоставил по-богати възможности за тестване и асертации на резултатите.**

1. Какво е Софтуерна документация, за какво служи? Дайте примери.

Софтуерната документация е съвкупност от текстове, снимки, диаграми и други материали, които обясняват как да се инсталира, конфигурира, използва и поддържа даден софтуерен продукт или приложение. Тази документация е предназначена за потребителите на софтуера, разработчиците, тестващите и всички други лица, които са свързани с неговото използване или разработване.

Ето някои от основните цели и предназначения на софтуерната документация:

* **Инструкции за Инсталация и Конфигурация:** Тези документи описват стъпките за инсталиране на софтуерния продукт на компютърна система и неговата конфигурация за оптимално функциониране.
* **Ръководство за Потребителя:** Това е основният документ, който обяснява как да се използва софтуерът. Тук се описват интерфейсите на потребителите, функционалностите и как да се използват.
* **Документация за Администратора:** Администраторската документация се фокусира върху аспекти като управление на потребителите, сигурност, резервно копиране на данни и други административни функции.
* **Ръководство за Отстраняване на Проблеми и Поддръжка:** Тази част от документацията предоставя помощ при откриване и отстраняване на проблеми с софтуера, както и информация за поддръжка и обновяване на продукта.
* **Справочници за Лицензи и Ползване:** Тук се обясняват условията на лицензиране на софтуера и правата и задълженията на потребителите.
* **Техническа документация:** Описва архитектурата, дизайна и технологиите, използвани в разработката на софтуера. Тя обикновено е насочена към разработчиците и техните нужди за разбиране на вътрешната структура и функционирането на системата.
* **Потребителска документация:** Предоставя информация за начина на използване на софтуера от страна на крайните потребители. Тя може да включва инструкции за инсталация, ръководство за потребителя, туториали, FAQ и други материали, които помагат на потребителите да се ориентират и да използват софтуера ефективно.
* **API документация:** Документация за програмния интерфейс (API) на софтуера, която описва как да се използва API-то, какви заявки да се правят и как да се обработват отговорите.

Софтуерната документация е важен инструмент за успешното внедряване, използване и развитие на софтуерни продукти. Тя улеснява потребителите в разбирането на продукта, намалява времето за обучение, помага за отстраняване на проблеми и предоставя необходимите ресурси за разработка и интеграция на софтуера в различни системи.

**Използвана литература/Източници**

<https://softuni.bg/blog/unit-testing-what-you-need-to-know>

<https://www.zaptest.com/bg/%D0%BA%D0%B0%D0%BA%D0%B2%D0%BE-%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D1%8F%D0%B2%D0%B0-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D1%82%D0%BE-%D1%82%D0%B5>

<https://medium.com/smallcase-engineering/testing-fundamentals-mocking-11cc5301df01>

<https://www.zaptest.com/bg/%D0%BA%D0%B0%D0%BA%D0%B2%D0%BE-%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D1%8F%D0%B2%D0%B0-%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D1%82%D0%BE>