Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка архітектури АРМ працівника амбулаторії

Виконав(ла): студент(ка) 4 курсу, групи СПс-41
спеціальності 121 Програмна інженерія
(шифр і назва спеціальності)

(підпис) Дронга П.О. (прізвище та ініціали)

Керівник (підпис) Яворська Є.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Програмна інженерія
(шифр і назва спеціальності)

студенту Дронзі Павлу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка архітектури АРМ працівника амбулаторії

Керівник роботи Яворська Євгенія Богданівна, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «_15__» *квітня* 2024 року № *4/7-343*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Вимоги замовника, технічні умови, технічне завдання

4. Зміст роботи (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Гурик О.Я., доцент кафедри МТ		
основи охорони праці	Лазарюк В.В., доцент кафедри МТ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз завдання		
3	Виконання розділу 1		
4	Виконання розділу 2		
5	Виконання розділу 3		
7	Оформлення пояснювальної записки		
8	Оформлення графічного та презентаційного матеріалу		
9	Попередній захист		
10	Захист		

Студент _____ Дронга П.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Яворська Є.Б.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра на тему «Розробка архітектури АРМ працівника амбулаторії» написана Дронгою Павлом Олеговичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-41

Кваліфікаційна робота бакалавра містить:

с. 55, рис. 10, табл. 3, джер. 28

Реалізується програма на основі MS Access для автоматизованого робочого місця лікаря невролога на основі методу Agile Scrum. Проводиться аналіз вимог, що дозволяє сформувати беклог продукту, а також задати та визначити беклог спринтів. Ідентифіковані ключові бізнес-процеси лікаря невролога проектуються у нотації UML Activity Diagram з різним рівнем деталізації, крім того, ведеться завдання архітектури даних. Змодельовані процеси та дані реалізуються у середовищі MS Access. Проводяться функціональні та інтеграційні види випробувань розробленого додатку.

Ключові слова: гнучка agile scrum, agile scrum проект, agile scrum розробка, арм, бізнес процес, діаграма діяльності UML, діаграма активності UML Actvity Diagram, діаграма класів UML Class Diagram.

ABSTRACT

A program based on MS Access is implemented for the automated workplace of a neurologist based on the Agile Scrum method. An analysis of requirements is carried out, which allows forming the product backlog, as well as setting and defining the backlog of sprints. The identified key business processes of the neurologist are projected in the UML Activity Diagram notation with different levels of detail, in addition, the data architecture task is conducted. Simulated processes and data are implemented in the MS Access environment. Functional and integration tests of the developed application are conducted.

Keywords: flexible agile scrum, agile scrum project, agile scrum development, arm, business process, UML activity diagram, UML Activity Diagram, UML Class Diagram.

ЗМІСТ

ВСТУП

За останні роки комп'ютерні технології проникли в багато сфер людської діяльності, в тому числі і в медицину. Велика кількість функціональних можливостей комп'ютерної техніки дає безмежні напрямки їх застосування, а можливість підвищити ефективність роботи лікаря робить ПК незамінним помічником у лікуванні.

Лікарям як користувачам, які мають конкретні предметні області медичних знань, потрібні максимально прості, зручні та продуктивні програмно-технічні засоби інформаційного сервісу. Важливо, щоб автоматизоване робоче місце (АРМ) не порушувало звичного для лікаря ритму та стилю роботи.

Проблеми, з якими стикається лікар на амбулаторному прийомі, це насамперед нестача часу. Розробка медичної бази даних дозволить вирішити цю проблему, оскільки може надати поточні відомості про стан здоров'я. У результаті лікар зможе більше часу проводити з пацієнтом, а, отже, більше часу приділяти на діагностику патології, що в свою чергу дозволить своєчасно призначити необхідне і правильне лікування.

Обсяг професійних знань, які необхідні успішної роботи лікаря, дуже великий. Зараз він зростає і продовжує постійно зростати. Тому, на сьогоднішній день автоматизація робочого місця лікаря є значним фактором в оперативності лікаря. Цього можна досягти за рахунок розробки програми за допомогою систем керування базами даних (СУБД), наприклад, MS Access. Ця програма сумісна з операційною системою Microsoft Windows, яка використовується на більшості комп'ютерів медичних установ, що важливо для зручності використання.

АРМ лікаря призначене не тільки для ведення історій хвороби, їх зберігання та складання звітів. Його особливе та найважливіше призначення – надавати постійну та різнобічну допомогу лікарю у його міркуваннях та рішеннях, нагадувати та підказувати, страхувати від низки помилок, надавати інформацію для роздумів.

Метою даної є розробка програмного комплексу для використання на робочому місці працівника амбулаторії (лікар-невролог) у вигляді автоматизованої системи. Програмне забезпечення уможливить збереження інформації про пацієнта та його візит для подальшого динамічного спостереження. Отже, в даному випадку людський фактор не повинен працювати, а значить, втрата даних про пацієнта знижується в кілька разів. Таким чином, щоб досягти вищезазначеної мети, нам потрібно реалізувати наступні завдання:

- описати та застосувати методологію впровадження ІС за допомогою методології Agile Scrum;
- ідентифікувати вимоги до програми;
- спроектувати процеси, дані та програми;
- розробити додаток у середовищі MS Access;
- провести тестування.

Розроблена під час кваліфікаційної роботи база даних дозволить упорядкувати та систематизувати роботу працівника амбелаторії. А це в свою чергу може вплинути на продуктивність праці користувача, який не буде витрачати час на пошук необхідної інформації про пацієнта.

1 ОПИС МОДЕЛІ ВПРОВАДЖЕННЯ ПРОГРАМНИХ ПРОДУКТІВ AGILE SCRUM

1.1 Гнучка методологія розробки Agile

Розробка програмного забезпечення – це праця, яка потребує своєчасного ухвалення правильних рішень. Але всі ухвалені рішення потрібно якось «упорядкувати». Сьогодні однією з цінних якостей керівника при побудові сучасного бізнес-процесу вважається вміння керувати командами, які працюють над проектом загалом і окремими продуктами. Для цього потрібен особливий гнучкий підхід, довіра колегам та готовність до змін ринку. Принципи гнучкого управління включені до концепції Srum та Agile, розробленої компаніями, зайнятими у сфері IT [22, 24, 26].

Agile – це сімейство "гнучких" підходів до розробки програмного забезпечення. Згідно з цим підходом, проект розбивається не на послідовні фази, а на маленькі підпроекти, які потім «збираються» у готовий продукт. Таким чином, верхньорівневе планування проводяться для проекту, а наступні етапи: розробка, тестування та інші проводяться для кожного міні-проекту окремо. Agile виник у IT-середовищі, але потім поширився і на інші сфери – від промислової інженерії до штучного інтелекту.

Agile припускає, що при реалізації проекту не потрібно спиратися лише на заздалегідь створені докладні плани. Важливо орієнтуватися на умови зовнішнього і внутрішнього середовища, що постійно змінюються, і враховувати зворотний зв'язок від замовників і користувачів. Це заохочує розробників та інженерів експериментувати та шукати нові рішення, не обмежуючи себе жорсткими рамками та стандартами.

1.2 Ідентифікація вимог

Ідентифікація попередніх вимог до нового продукту – це те, що дозволяє визначити характеристики нового продукту відповідно до вимог клієнта. Команда, яка займається проектом, ставить певні цілі, відповідні або якимось рис продукту конкурента, або вимогам ринку, або ідентифікують якісь ознаки, які кінцевий користувач хотів би додати до вже існуючого продукту.

Проблеми, які доводиться вирішувати спеціалістам у процесі створення програмного забезпечення, не завжди зрозумілі. Зокрема, важко чітко описати ті дії, які має виконувати система. Опис функціональних можливостей та обмежень, що накладаються на систему, називається вимогами до цієї системи. Вимоги до продукту повинні бути встановлені таким чином, що могло б гарантувати їхню адекватність та правильний переклад з мови користувача. У першу чергу необхідно виявити всі можливі вимоги, що пред'являються до програмного продукту, що розробляється [22, 24, 26, 27].

Виявити їх можна за допомогою аналізу вимог - процесу збирання вимог до програмного забезпечення (ПЗ), їх систематизації, документування, аналізу, виявлення протиріч, неповноти, вирішення конфліктів у процесі розробки програмного забезпечення. У процесі збору вимог важливо брати до уваги можливі протиріччя вимог різних заінтересованих осіб, таких як замовники, розробники чи користувачі. Аналіз вимог включає такі кроки:

- збір вимог: спілкування з клієнтами та користувачами, щоб визначити, які їхні вимоги;
- аналіз вимог: визначення, чи є зібрані вимоги неясними, неповними, неоднозначними, чи суперечать і потім вирішення цих проблем;
- документування вимог: вимоги можуть бути задокументовані в різних формах, таких як простий опис, сценарії використання, власні історії, або специфікації процесів.

1.2.1. Методологія Scrum.

Окремим Agile-підходом є Scrum - "підхід структури". Тобто це набір принципів, на яких будується процес розробки, що дозволяє жорстко фіксовані і невеликі за часом цикли – спринти (Sprints), надавати кінцевому користувачеві працююче ПЗ з новими можливостями, для яких визначено найбільший пріоритет. Можливості ПЗ до реалізації в черговому спринті визначаються на початку спринту на етапі планування і не можуть змінюватися на його протязі. При цьому суворо фіксована невелика тривалість спринту надає процесу розробки передбачуваність та гнучкість. В основі методології Scrum лежить проста ідея. Коли б не було запущено проект, вам ніщо не заважає регулярно перевіряти хід робіт і послідовно з'ясовувати: чи справляєтеся ви із завданням; чи в потрібному напрямку рухаєтеся; Чи створюєте саме те, що насправді хоче отримати замовник. Вам також ніщо не заважає постійно порушувати такі питання: чи є способи удосконалити методи розробки та виконувати роботу найбільш якісно та швидко; чи існують чинники, що перешкоджають вашим завданням [1].

Підхід, що лежить в основі методики Scrum, можна застосовувати в різних видах діяльності, які потребують колективної роботи. Важливими характеристиками Scrum є її гнучкість та орієнтованість на клієнта, оскільки вона передбачає його (клієнта) безпосередню участь у процесі роботи. Scrum не вимагає впровадження будь-яких дорогих інструментів. Схему методики Scrum коротко можна описати в такий спосіб:

- насамперед треба вибрати «Власника продукту» (Product owner) – людину, здатну бачити те, що ви збираєтеся створити або досягти. Вона є сполучною ланкою між командою розробки та замовником. Її завданням є максимальне збільшення цінності продукту, що розробляється, і роботи команди;
- потім потрібно зібрати "Команду розробки" (Development team), до якої увійдуть люди, які безпосередньо виконують роботу. Вони повинні мати навички та знання, які допоможуть втілити ідею власника продукту в життя. А також слід мати такі якості та характеристики, як самоорганізація та багатофункціональність;

- далі необхідно вибрати «Скрам-майстра» (Scrum master) - того, хто стежитиме за ходом реалізації проекту та допомагатиме команді усувати перешкоди на шляху досягнення мети;
- приступаючи до роботи, потрібно створити максимально повний список усіх вимог, що пред'являються до продукту або мети, упорядкований за ступенем важливості, що підлягають реалізації. Пункти цього списку мають бути розставлені за пріоритетом. Список має назву «Беклог продукту» (Product Backlog). Він може розвиватися та змінюватися протягом усього терміну реалізації проекту;
- учасники команди повинні оцінити за своєю системою оцінок кожен пункт щодо складності та витрат, які будуть потрібні для його виконання. Потім учасники, скрам-майстер та власник продукту повинні провести перші скрам-збори, на яких вони запланують спринт - певний час для виконання частини завдань. Тривалість його має перевищувати один місяць. Тим не менш, вважається, що чим коротшим є спринт, тим гнучкішим є процес розробки, релізи виходять частіше, швидше надходять відгуки від споживача, менше часу витрачається на роботу в неправильному напрямку. З іншого боку, при більш тривалих спринтах команда має більше часу на вирішення проблем, що виникли в процесі, а власник проекту зменшує витрати на наради, демонстрації продукту і т. п. Буває таке, що доводиться зупиняти спринт, але це відбувається у виняткових ситуаціях. Спринт може бути зупинений до закінчення відведених днів, або його може зупинити команда, якщо розуміє, що не може досягти мети спринту у відведений час. Спринт може зупинити власник продукту, якщо необхідність досягнення мети спринту зникла;
- команда повинна постійно прагнути до того, щоб перевершити в новому спринті свої власні результати за попередній спринт;
- після завершення спринту команда робить його огляд - проводить зустріч, де учасники розповідають, що зроблено за спринт;

- після показу результатів роботи за спринт учасники проводять ретроспективні збори, де обговорюють, що команда робила добре, що можна зробити краще, що можна покращити прямо зараз.

Переваги такої методології, як уже говорилося, у її гнучкості та адаптивності. Завжди можна щось змінити у продукті, додати ще одну функцію. Agile Scrum дуже зручний, коли замовник сам остаточно не знає, чого хоче. Таким чином виходить випустити програму з основними функціями, а з кожним наступним спринтом додавати до неї нові.

Ще один плюс цієї методології – це самостійність та самоорганізованість кожного учасника проекту. Але в такому випадку досить багато уваги приділяється добору персоналу. Особливістю Agile Scrum є залучення до процесу всіх учасників команди, причому у кожного учасника є своя певна роль. Команда самостійно регулює власну роботу, встановлює тимчасові рамки та умови роботи у межах спринту. Керівник проекту, який є майстром Scrum, неспроможна вказувати команді, як створювати продукт.

При цьому в команду розробки входять функціональні фахівці, які мають необхідні навички для роботи. У структурі команди немає поділу підрозділи, виконують окремі функції. Навіть якщо окремі члени команди володіють вузькоспеціалізованими знаннями в різних галузях, відповідальність за продукт лежить на всій команді загалом. Проте є така проблема, як невизначеність. Кількість спринтів необмежена, тому складно поставити кінцеву дату у проекті. Таким чином, можна виділити плюси Agile Scrum:

- швидкий запуск;
- швидке реагування на зміни - гнучкість та адаптивність;
- зв'язок у реальному часі між командою та клієнтом,

а також мінуси:

- вимагає висококваліфікованих та мотивованих фахівців;
- вимагає багато часу від клієнта;
- відсутність довгострокових планів.

1.2.2. Користувальницькі та функціональні вимоги, їхня пріоритизація.

Таким чином, насамперед було проведено роботу із замовником системи. Невролог профодить профілактику, досліджує та лікує хвороби нервової системи. Його основним завданням є виявити ураження саме нервової системи (якщо є), а також можливі причини з боку інших систем та органів. При цьому зазначені скарги можуть бути вторинними, первинна причина не неврологічного характеру (наприклад, оніміння пальців рук і ніг при діабеті або отруєння токсичними речовинами).

Після цього було складено список виявлених вимог користувача (вимоги, що формулюються користувачами до кінцевого продукту), на які буде покладатися розробка програми [22, 23, 27]:

- можливість зберігання даних (про пацієнта, про лікування пацієнта, відвідування пацієнтів);
- можливість перегляду даних із таблиці «Загальний огляд»;
- можливість перегляду даних із таблиці «Лікування пацієнта»;
- можливість перегляду даних із таблиці «Особисті дані пацієнта»;
- можливість перегляду даних із таблиці «Відвідування»;
- можливість редагування даних із таблиці «Загальний огляд»;
- можливість редагування даних із таблиці «Лікування пацієнта»;
- можливість редагування даних із таблиці «Особисті дані пацієнта»;
- можливість редагування даних із таблиці «Відвідування»;
- можливість заводити інформацію про пацієнтів у структуровані електронні записи;
- пошук усіх відвідувань конкретного пацієнта;
- можливість пошуку даних про конкретного пацієнта;
- можливість роботи із додатком на будь-якому ПК медичного закладу;
- можливість виведення інформації на паперовий носій;
- захист даних від несанкціонованого доступу;
- простота та зручність інтерфейсу.

Програмне забезпечення розроблятиметься у середовищі MSAccess. Інтерфейс даної програми простий і зручний, він дозволяє створювати власну БД та додаток для неї. Дані зберігаються в одному файлі і розприділяє їх по різним таблицям. Таким чином, доступна інформація буде легко надаватися кінцевому користувачеві. Всі версії Access мають у своєму арсеналі засоби, що значно спрощують введення та обробку даних, пошук даних та надання інформації у вигляді таблиць, графіків та звітів.

Дуже важливим етапом роботи з беклог продукту є вибудовування вимог у порядку їх пріоритету. Необхідно визначити, в якій послідовності буде проходити реалізація завдань, що буде випущено в першій версії продукту, що може бути включено до наступних поставок, а що може бути залишено на той випадок, якщо залишиться час. Провести сортування бэклога і побудувати той чи інший порядок завдань можна базуючись на різних умовах, виходячи з різних цінностей. Існує різноманітні варіанти технік, за допомогою яких можна провести сортування беклогу, однією з них є методика пріорітизації MoSCoW.

Оцінюючи пріоритету методом «MoSCoW» кожної задачі підбирається найбільш відповідне їй одне твердження з чотирьох. Кожне твердження є індикатором певного рівня важливості завдання. Яке із тверджень більше відповідає дійсності, до такої групи пріоритету і належить дана історія користувача:

- Must have this – це обов'язково має бути. "Must" вимоги не підлягають обговоренню і в будь-якому випадку мають бути реалізовані. Якщо вони не поставлені користувачеві в найближчих релізах, весь проект не має сенсу.
- Should have this if at all possible – слід зробити, якщо це можливо. Дані завдання також важливі для користувача або замовника, однак, терміни їх поставки не такі критичні, як для «Must». До цієї категорії належать вимоги, які для функціоналу є критичними, а для поточного релізу – ні.
- Could have this if it does not affect anything else – можна зробити, якщо це вплине щось інше. Ця категорія вимог, яку бажано включити, але не впливає на реліз успіху.

- Will not have this time but would like in the future – зараз на це немає часу, але хотілося б зробити в майбутньому. Для цієї категорії ключовим моментом є те, що ці вимоги можуть бути також важливими (також, як і «Must»), однак їх можна залишити для пізніших поставок продукту.

Використовуючи цей метод, можна визначити пріоритети:

- Must have this – вимога 1. Оскільки саме ця вимога є основою для всього програмного забезпечення.
- Should have this if at all possible – вимоги 5-6. Це ті вимоги, що дозволяють виконувати безпосередньо роботу лікаря.
- Could have this if it does not affect anything else – вимоги 2-4. У Microsoft Office Access можна створити форми, які дозволяють переглядати дані. Це виглядає зручніше та приємніше, але переглянути дані можна і просто в таблицях, відкривши вікно редагування.
- Will not have this time but would like in the future – вимога 8. Цю вимогу можна залишити потім.

Наступним кроком виявимо функціональні вимоги (визначають функціональність (поведінку) програмної системи, щоб виконати вимоги користувача):

- база даних, що містить таблиці "Особисті дані пацієнтів", "Лікування пацієнтів", "Відвідування", "Загальний огляд";
- виведення на екран дані з таблиці "Особисті дані пацієнтів";
- виведення на екран дані з таблиці "Лікування пацієнта";
- виведення на екран дані з таблиці "Загальний огляд";
- виведення на екран даних із таблиці «Відвідування»;
- додавання даних про пацієнта;
- редагування даних про пацієнта у таблиці «Особисті дані пацієнтів»;
- редагування даних про пацієнта у таблиці «Загальний огляд»;
- редагування даних про пацієнта у таблиці «Лікування пацієнта»;
- пошук конкретного пацієнта з ПІБ;
- пошук конкретного пацієнта із таблиці «Відвідування»;

- друк звіту, що містить особисті дані пацієнта, загальний огляд та відвідування;
- програма має коректно працювати на всіх комп'ютерах медустанови;
- дані зберігаються безпосередньо у створюваному додатку;
- встановлення пароля на саму базу даних;
- доступність (легкість сприйняття).

1.2.3. Беклог продукту.

Беклог продукту, як говорилося раніше, є списком планованих поліпшень продукту, розставлених за пріоритетом. Він може мати практично будь-яким чином, у цій роботі цей перелік буде у вигляді таблиці [22, 23, 25].

Таблиця 1.1. Беклог продукту

№	Користувальницька вимога	Пріоритет MuSCoW	Функціональна вимога	Програмний компонент
1	Збереження даних	Must have	Зберігання інформації про пацієнтів, їх лікування, відвідування лікаря, про типи обстеження, про препарати	Програми, що працюють із базами даних «Особисті дані пацієнта», «Лікування пацієнта», «Огляд пацієнта», «Типи обстеження», «Препарати», «Відвідування»
2	Можливість пошуку даних певної інформації про конкретного пацієнта	Should have	Пошук конкретного пацієнта з ПІБ	Програма для пошуку
3	Можливість перегляду всіх відвідувань конкретного пацієнта	Should have	Пошук конкретного пацієнта із таблиці «Відвідування»	Програма для пошуку

Продовження табл. 1.1

4	Редагування даних про пацієнта у таблиці «Особисті дані пацієнтів»	Should have	Функція зміни інформації	Програма зміни інформації
5	Редагування даних про пацієнта у таблиці «Загальний огляд»	Should have	Функція зміни інформації	Програма зміни інформації
6	Можливість заводити інформацію про пацієнтів	Should have	Функція додавання даних про пацієнта	Програма для додавання інформації
7	Можливість роботи з додатком на будь-якому комп'ютері медичного закладу	Should have	Програма має коректно працювати на всіх комп'ютерах медустанови	Середовище розробки MS Access
8	Можливість виведення інформації на паперовий носій	Could have	Друк звіту, що містить особисті дані пацієнта, загальний огляд та відвідування	Програма для друку
9	Можливість перегляду особистих даних пацієнта	Could have	Ведення таблиці даних «Особисті дані пацієнта»	Програма з ведення особистих даних пацієнта
10	Можливість перегляду даних про лікування пацієнта	Could have	Ведення таблиці «Лікування пацієнта», інформація про виписані препарати, типи обстеження та діагнозу	Програма ведення даних про лікування пацієнта
11	Можливість внесення змін інформації	Could have	Функція зміни інформації	Програма зміни інформації
12	Захист інформації від несанкціонованого доступу	Could have	Встановлення пароля на саму базу даних	Програмне забезпечення
13	Можливість перегляду всіх відвідувань усіх пацієнтів	Could have	Ведення таблиці «Відвідування», що містить інформацію про дату відвідування та час відвідування	Програма ведення даних про відвідування
14	Простота та легкість інтерфейсу	Won`t have	Доступність (легкість сприйняття)	ПЗ, що розробляється

1.2.4. Завдання циклів розробки згідно з Agile Scrum.

Застосовуючи вищезгадану інформацію, потрібно створити максимально повний список усіх вимог, що висуваються до продукту, тобто створюємо беклог. Цей список складатиметься з користувацьких та функціональних вимог. Розробка програмного забезпечення проводитиметься таким чином:

Початок:

- аналіз вимог;
- моделювання ключового бізнес-процесу за допомогою нотації UML AD.

Перший спринт:

- зберігання інформації про пацієнтів, їх лікування, про діагнози, симптоми, про відвідування лікаря, про типи обстеження, про препарати. Для цього створимо необхідні таблиці у певному середовищі розробки;
- тестування реалізованих можливостей програми;
- демонстрація прототипу програми замовнику.

Другий спринт:

- можливість перегляду особистих даних пацієнта, тобто ведення певної таблиці (або таблиць), що містить інформацію про загальний огляд, типи обстеження; можливість зміни внесення інформації;
- тестування реалізованих можливостей програми;
- демонстрація прототипу програми замовнику.

Третій спринт:

- можливість пошуку даних певної інформації про пацієнта та можливість друку інформації на паперовий носій;
- тестування реалізованих можливостей програми;
- демонстрація прототипу програми замовнику.

Четвертий спринт:

- створення простого та легкого інтерфейсу; захист від несанкціонованого входу;
- виправлення помилок;

- тестування реалізованих можливостей програми;
- демонстрація прототипу програми замовнику.

1.3 Проектування ключових бізнес-процесів

Головним принципом розвитку справи є управління, побудоване на бізнес – процесах. При процесному управлінні ключові показники ефективності кардинально покращуються [2]. Бізнес-процес – це стійка цілеспрямована послідовність виконання функцій, спрямовану створення результату, має цінність споживача [3].

Модель бізнес-процесу формує єдину картину та бачення ситуації співробітників та керівництва підприємства. Цей метод дозволяє дати вартісну оцінку кожному окремому процесу всім бізнес-процесам організації в сукупності. При проектуванні ключових бізнес-процесів використовують дві моделі: AS-IS (як є) та TO-BE (як буде).

Перш ніж створити власну інформаційну систему, необхідно проаналізувати, як працює система в даний час. Тому для опису роботи проекту необхідно побудувати функціональну модель AS-IS. Аналіз цієї функціональної моделі дозволяє зрозуміти, де знаходяться найслабші місця, в чому полягатимуть переваги нових бізнес-процесів. Деталізація бізнес-процесів дозволяє виявити недоліки. Побудова моделі AS-IS допомагає точно зафіксувати, які процеси здійснюються на підприємстві на даний момент, які інформаційні об'єкти використовуються для виконання функцій різного рівня конкретизації. Ця модель дозволяє з'ясувати, «що» та «як» робити зараз перш, ніж визначити «що» та «як» робитиметься завтра.

Знайдені у моделі AS-IS недоліки можна виправити під час створення моделі TO-BE. Ця модель від попередньої відрізнятиметься не кардинально, а лише у невеликій своїй частині, тобто у якихось нюансах. Модель TO-BE

потрібна для оцінки наслідків впровадження інформаційної системи та аналізу альтернативних шляхів виконання роботи та документування того, як система функціонуватиме в майбутньому. За підсумком прототип і потім кінцевий варіант інформаційної системи будується на основі розробленої моделі TO BE.

1.3.1. Проектування на основі UML Activity Diagram.

Для моделювння поведінки системи потрібно уявити процес зміни її станів та конкретизувати особливості алгоритмічної та логічної реалізації операцій , які вона виконує. Для цього застосовують блок-схему чи структурну схему. Кожна покаазує послідовність виконання певних дій чи елементарних операцій, які уможливлять отримати бажаний результат.

На схемі бізнес-процесу вузли процесу можна зображати різними образами. Спосіб зображення вузлів та переходів важливий, оскільки від цього залежить легкість чи складність розуміння бізнес-процесу людьми. Узгоджені набори графічних елементів, у тому числі будуються схеми бізнес-процесів, називаються графічними нотаціями зображення бізнес-процесів. У цій роботі було обрано UML (Universal Modelling Language) – універсальна мова моделювання графічного опису, призначена для об'єктного моделювання в галузі розробки програмного забезпечення.

Моделюючи процес виконання операцій на UML використовують діаграму діяльності (Activity diagram).

UML AD нотація є простою для вивчення непідготовленим користувачем, вона інтуїтивно зрозуміла. Ця нотація використовує широко відомі графічні елементи. Наприклад, у ній для вибору одного з кількох напрямків використовується ромбік. А паралельно виконуються вузли-дії в цій нотації з'єднані з елементами - поділами-злиттями паралельними лініями, що відповідає діям, що одночасно виконуються. У UML AD нотації зображення процесів дуже схоже на блок-схеми, тобто багатьом російським користувачам зображення в UML AD нотації відразу підсвідомо зрозумілі.

1.3.2. Проектування процесів у моделі AS-IS за допомогою UML AD.

На рисунку розглядається перший рівень проектування процесу роботи лікаря-невролога в інструкції UML AD моделі «AS-IS», яка передбачає проектування процесів в даний момент часу. На даному етапі проектування описуються основний та допоміжний бізнес-процеси лікаря-невролога.

Рисунок 1.1 – Подання процесу в UML AD на першому рівні моделі «AS-IS»

Щоб створити програмне забезпечення, потрібно більше інформації, тому необхідно провести детальніше проектування. З другого краю рівні розглядається деталізований процес «Прийняти пацієнта». Джерелом інформації служить сам пацієнт та його документи.

У моделі TO-BE, якщо порівняти наступні рисунки, можна помітити, що замість медичної картки на паперовому носії, напрямків та довідок, можна використовувати електронну базу даних. До неї кожен співробітник може вносити необхідну інформацію про пацієнта: особисту інформацію, скарги на стан здоров'я, результати огляду лікарем.

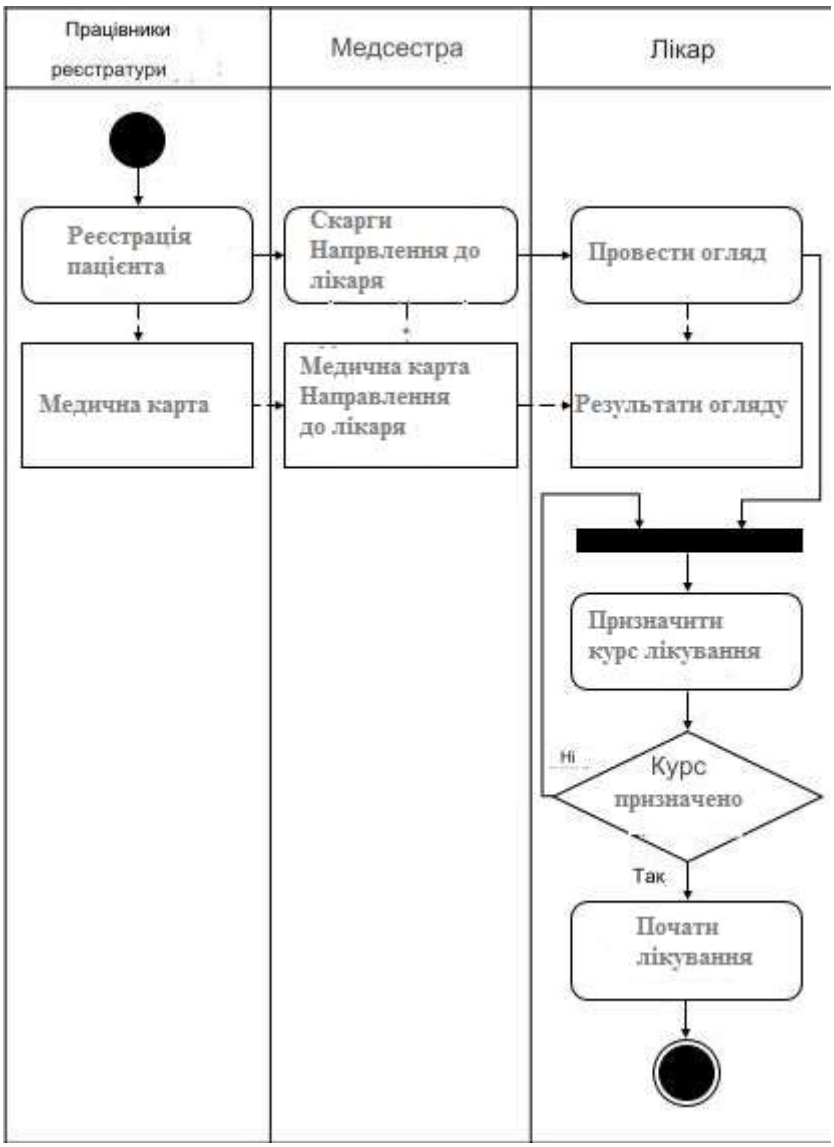
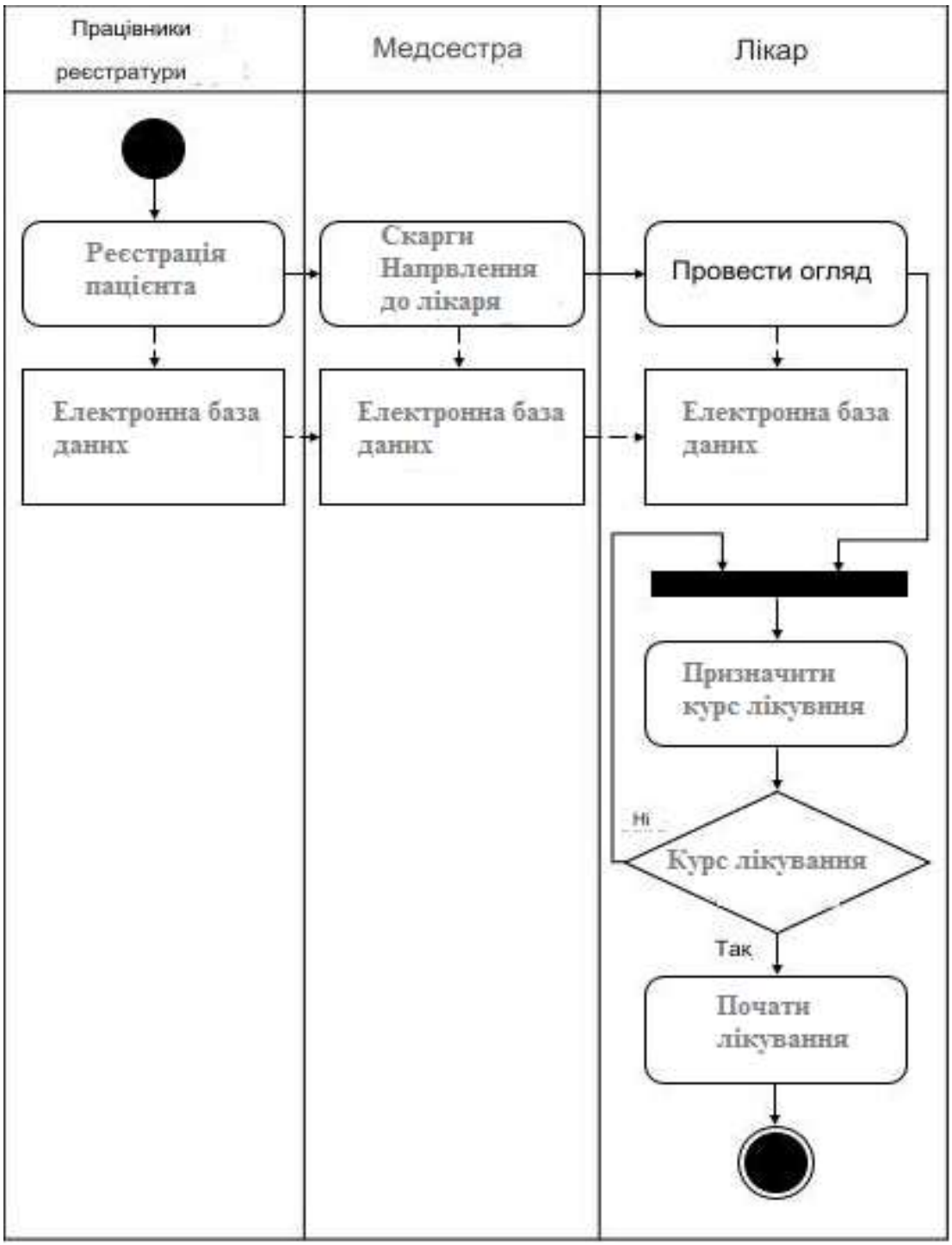


Рисунок 1.2 – Подання процесу в UML AD на другому рівні в моделі «AS-IS»



Рисуноку 1.3 – Подання процесу в UML AD на другому рівні моделі «TO-BE»

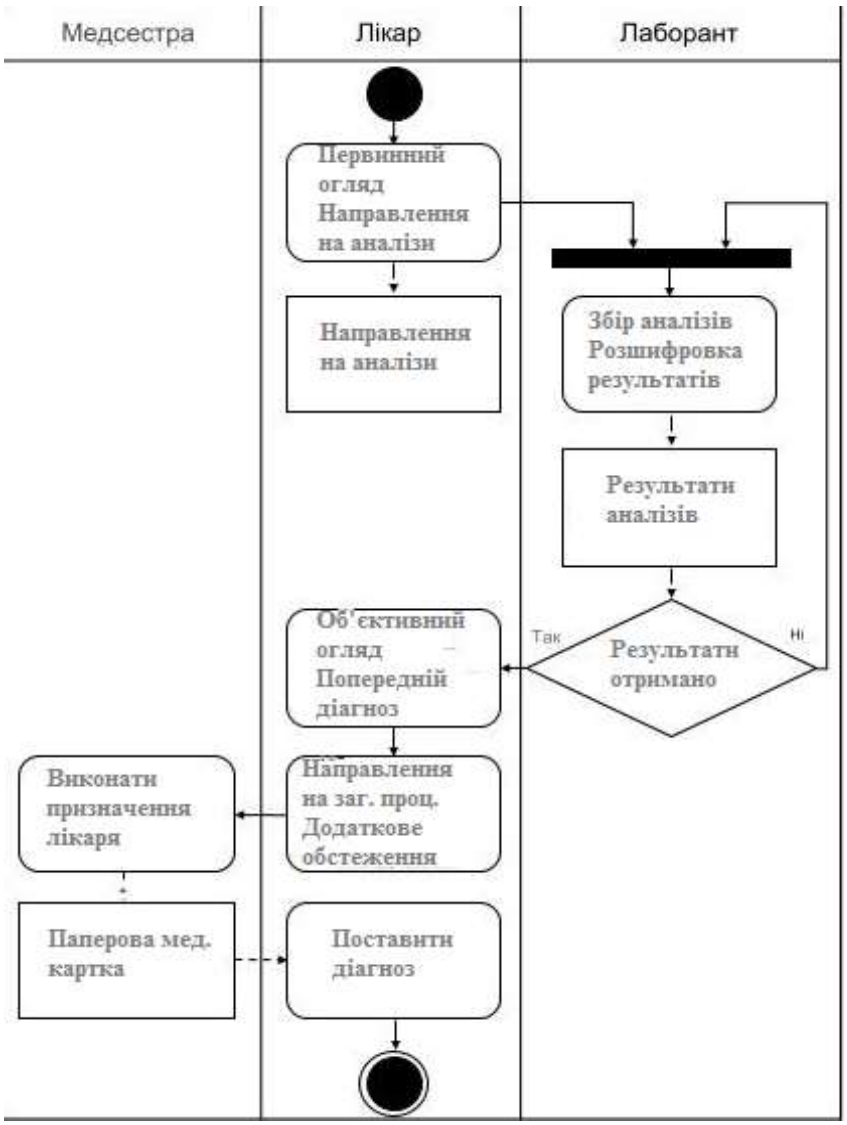


Рисунок 1.4 – Подання процесу в UML AD на третьому рівні моделі «AS-IS»

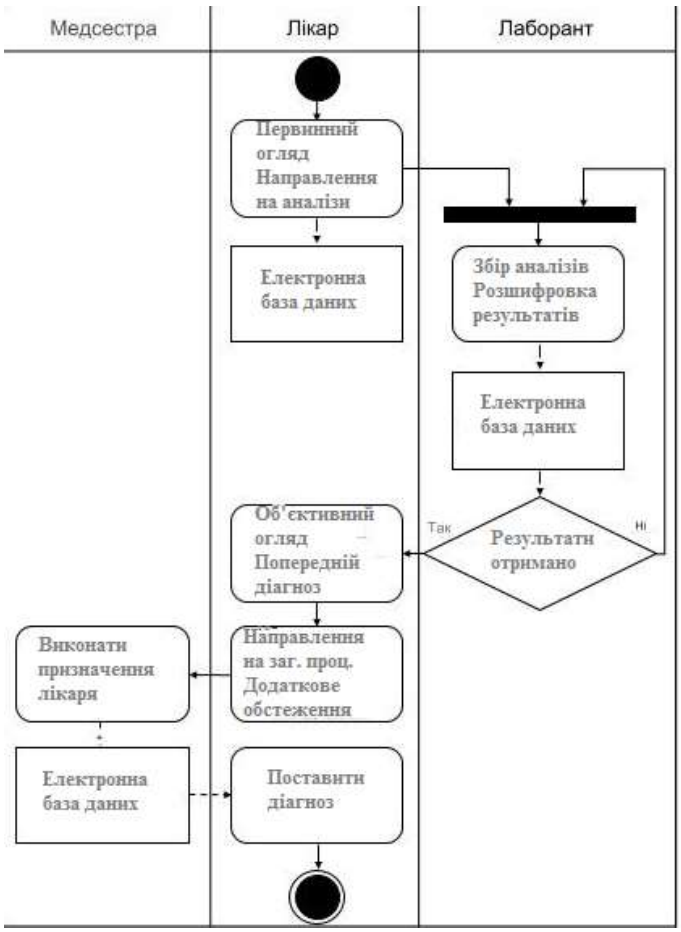


Рисунок 1.5 – Подання процесу в UML AD на третьому рівні моделі «TO-BE»

При декомпозиції все більш наочна стає роль програми, що розробляється. Для точного виявлення автоматизації потрібно розглянути наступний рівень. На третьому рівні розглядається процес «Провести огляд». Цей процес поділяється на кілька етапів, що потрібно для того, щоб поставити точний діагноз та призначити правильне лікування пацієнтові.

З результатів моделювання видно, що це паперові носії повністю замінені базою даних. Ця автоматизація досить сильно прискорює пошук інформації про пацієнта лікарями та внесення записів про стан хворого, а отже, сам лікар може більше часу приділити безпосередньо на огляд та лікування пацієнта.

1.4 Розробка програми

1.4.1 Прототип програми (перший спринт).

Реалізація програми на першому спринті є звичайною таблицею з елементами управління самого середовища розробки MS Access. Тут немає інтерфейсу, який міг би полегшити роботу користувача з програмою.

Після того, як всі необхідні таблиці були створені, ми створюємо схему та зв'язки всіх використовуваних таблиць даних, необхідних додатку. Схема даних будувалася шляхом додавання необхідних таблиць даних, виділенням їх ключових полів, необхідні створення зв'язків з-поміж них. Усе це виконувалося як «конструктора».

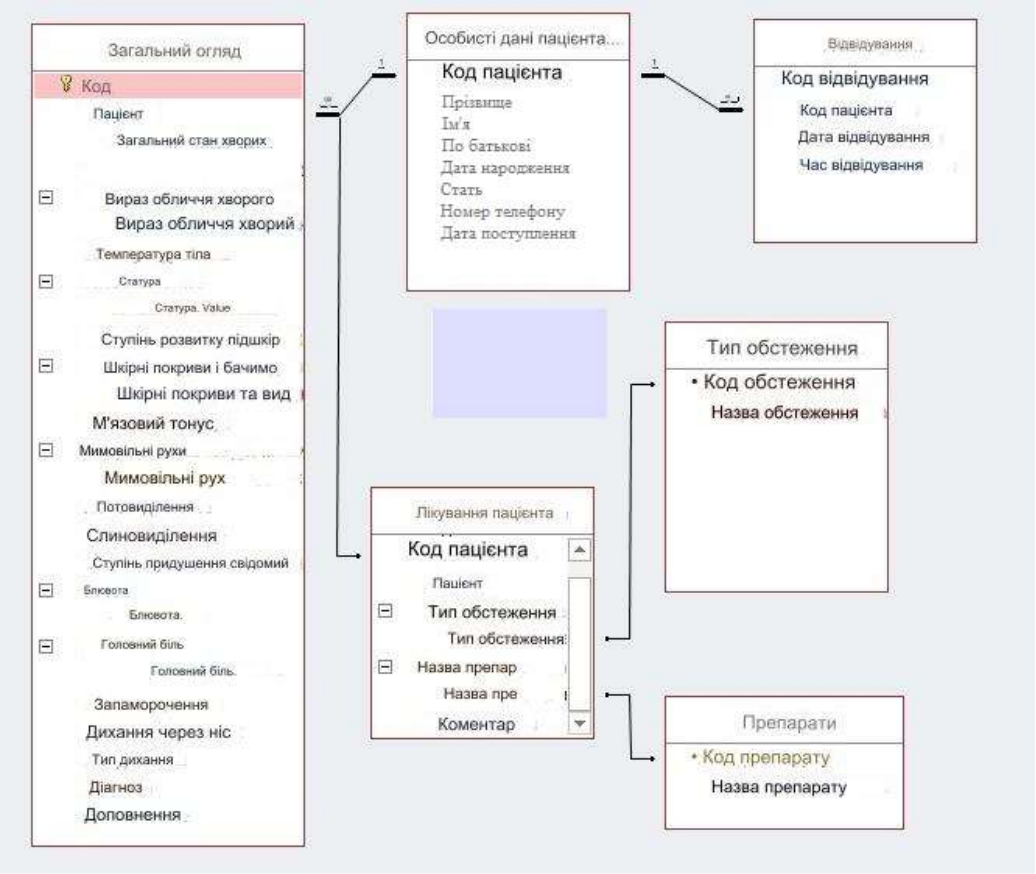


Рисунок 1.7 – Схема даних

1.4.2. Прототип програми (другий спринт).

На цьому етапі реалізовується можливість перегляду даних, і навіть можливість зміни внесення інформації. Для цього створюється «Форма» – об'єкт, який допомагає користувачу додавати, редагувати та відображати дані. Випадаючі списки, які видно при натисканні на стрілку, дозволяють змінювати інформацію.

1.4.3 Прототип програми (третій спринт).

Наступним кроком було створення запитів для пошуку потрібної інформації серед усієї бази даних. З їхньою допомогою проводиться відбір тих записів, які необхідні складання певних звітів чи форм, тобто робота здійснюється з частиною бази даних. Запит під назвою «Пошук пацієнта» дозволяє знайти конкретного пацієнта з таблиці «Лікування пацієнта».

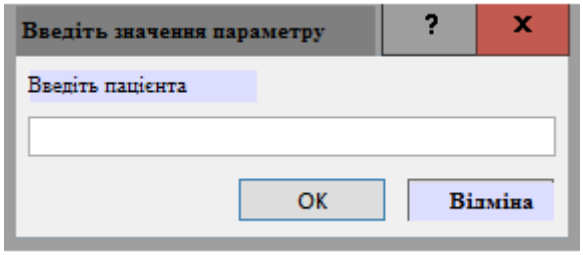


Рисунок 1.8 – Вікно пошуку відвідувань конкретного пацієнта

А подібний запит із назвою «Пошук усіх відвідувань пацієнтом» допомагає користувачеві відстежити всі дати та весь час відвідувань певного пацієнта. Наступним кроком було створення звітів. У Microsoft Access вони служать виведення на екран, чи друку інформації з бази даних.

1.4.4. Кінцева програма (четвертий спринт).

Четвертий спринт спрямований на розробку інтерфейсу та навігацію. Кінцевий інтерфейс розробленої програми виглядає так (див. рисунок).

Рисунок 1.9 – Головне меню програми

Головне меню складається із чотирьох кнопок. Перша - "Особисті дані пацієнта", при натисканні на неї з'являються також чотири кнопки: "Перегляд особистих даних пацієнтів", "Додати нового пацієнта", "Знайти пацієнта" та "Назад". При натисканні «Додати нового пацієнта» на екрані з'являється порожня форма, до якої можна додати всі особисті дані нового пацієнта.

Якщо натиснути на «Лікування», відкриється інтерфейс з наступними кнопками: «Загальний огляд», «Створити новий запис для загального огляду», «Переглянути призначене лікування», «Назад». Далі було додано кнопки навігації. «Стрілки» вліво та вправо дозволяють переходити до попереднього запису та наступного відповідно. Кнопка із зображенням «Лупи» має функцію пошуку, назва кнопки «Вихід» говорить сама за себе.

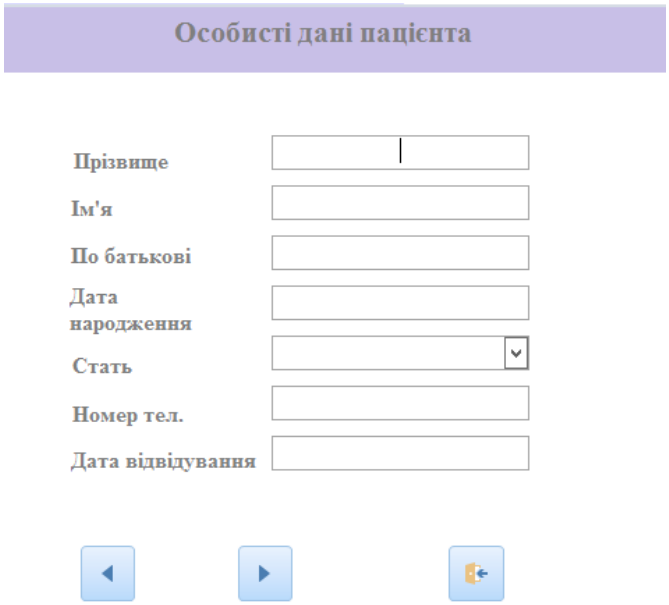


Рисунок 1.10 – Форма для додавання нового пацієнта

Останнім пунктом розробки цієї програми був захист від несанкціонованого доступу: під час запуску програми програма запитує пароль. Також були внесені виправлення як функціональні, так і візуальні у кінцеве меню інтерфейсу. У результаті отримано додаток з основним меню для лікаря-невролога, що містить 5 кнопок. Одна з кнопок - "Вихід", що дозволяє закрити програму.

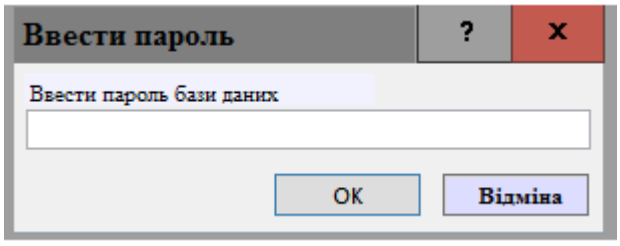


Рисунок 1.11 – Спливаюче вікно під час відкриття бази даних

2 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

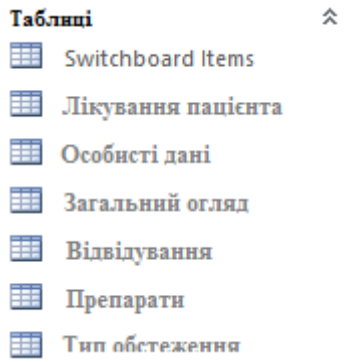
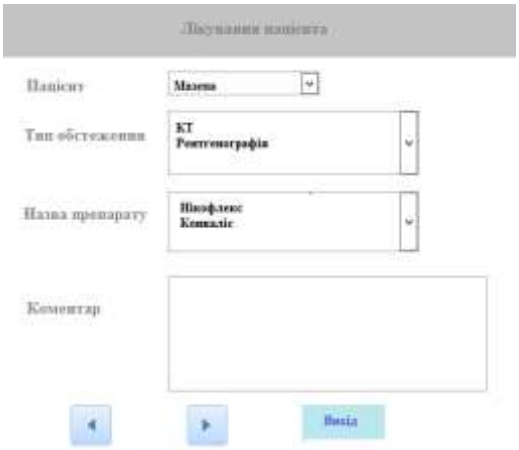
Тестування є невід'ємною частиною розробки програм. Це процедура перевірки відповідності між наявною та очікуваною поведінкою програми, яка проводиться на кінцевому, обраному певним чином, наборі тестів. У цій роботі при проектуванні програмного забезпечення було обрано методологію Agile Scrum, саме з цієї причини неможливо не згадати про Agile-тестування. Як було сказано на початку цієї роботи, Agile - гнучка методологія, що вимагає командної роботи з усіма учасниками проекту. У цій методиці кожен член команди несе відповідальність відповідно до своєї ролі та роботи, також протягом всієї розробки необхідні вимоги клієнта і, найголовніше, гнучкий підхід дозволяє вносити зміни майже на будь-якому етапі проектування. Це означає, що у Agile Scrum тестувати необхідно у кожному ітерації, що було зроблено протягом всієї роботи: тестування проводилося у кожному спринті.

2.1. Функціональне тестування

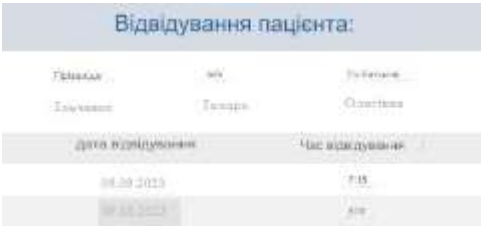
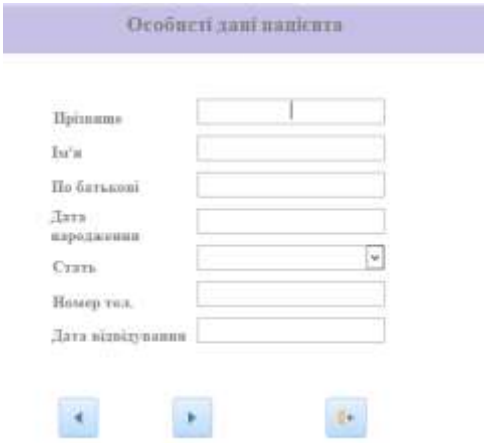
Тестування ПЗ – аналізування програмного продукту та супутньої документації з метою виявлення недоліків у роботі та підвищення його якості [17]. Зазвичай для перевірки працездатності розробленого ПЗ проводять три види тестування: функціональне, інтеграційне та навантажувальне.

Функціональне тестування спрямоване на перевірку коректності роботи функціональності докладання (коректність реалізації функціональних вимог).

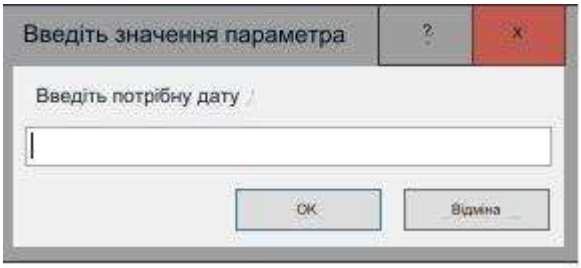
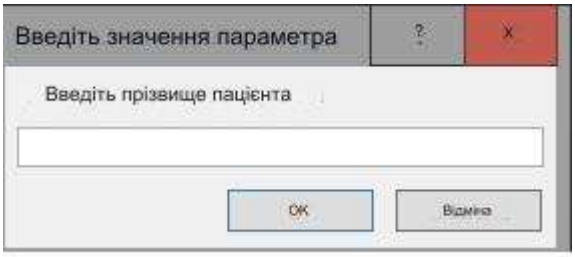
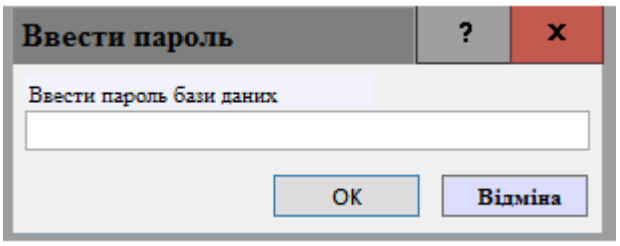
Таблиця 2.1 – Функціональне тестування

Функціональна вимога	**Програмний компонент**	**Графічне уявлення**
База даних, що містить таблиці "Особисті дані пацієнтів", "Лікування пацієнтів", "Відвідування", "Загальний огляд"	ПЗ, що розробляється	
Виведення на екран дані з таблиці "Особисті дані пацієнтів"	Програма з виведення даних на екран про особисті дані пацієнта	
Виведення на екран дані з таблиці "Лікування пацієнта"	Програма виведення даних на екран про лікування пацієнта	

Продовження табл. 2.1

Виведення на екран даних з таблиці «Загальний огляд»	Програма виведення на екран даних з таблиці «Загальний огляд»	
Виведення на екран даних з таблиці «Відвідування»	Програма з виведення даних на екран про відвідування пацієнтів	
Додавання даних про нового пацієнта	Програма із введення даних про нового пацієнта	
Редагування даних про пацієнта у таблиці «Особисті дані пацієнтів»	Засіб для зміни інформації	

Продовження табл. 2.1

Пошук конкретного пацієнта із таблиці «Відвідування»	Програма з виведення на екран даних, що запитуються	
Пошук конкретного пацієнта з ПІБ	Програма з виведення на екран даних, що запитуються	
Встановлення пароля на саму базу даних	ПЗ, що розробляється	

Таким чином, можна сказати, що функціональні вимоги були реалізовані та функціональне тестування пройшло успішно.

2.2 Тестування навантаження

Будь-яке програмне забезпечення має працювати під навантаженням тривалий час. Збої та відмови системи можуть призвести до збитків, втрати клієнтів та інших неприємних наслідків. Навантаження тестування дозволяє визначити, як і з якою швидкістю працює програма під певним навантаженням. За допомогою навантажувального тестування оцінюється відповідність продуктивності продукту вимогам, сформульованим у завданні. Для тестування

було вибрано таку кількість записів: 1, 10 та 100 (приблизна кількість пацієнтів за місяць). Дані щодо навантажувального тестування наведені у табл.5.2. На основі отриманих результатів можна сказати, що тестування навантаження пройшло успішно.

Таблиця 2.2 – Результати тесту навантаження

Кількість записів	**Пошук наявної інформації**	**Виведення на екран усієї наявної інформації**
1	0,10...0,15 сек.	0,10...0,15 сек.
10	0,12...0,24 сек.	0,11 … 0,23 сік.
100	0,28 … 0,42 сік.	0,26 … 0,45 сік.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Долікарська допомога при попаданні стороннього тіла у вухо, ніс, очі, дихальні шляхи, кишковий тракт.

3.1.1 Сторонні тіла носа

Долікарська допомога – це комплекс медичних заходів, спрямованих на надання медичної допомоги при невідкладних станах, які стаються на виробництві, у побуті, під час дорожньо-транспортних пригод, катастроф, техногенних аварій та при гострих неврологічних, терапевтичних, хірургічних та термінальних станах. Ненадання долікарської допомоги при нещасних випадках, раптових гострих захворюваннях людини призводить до тяжких наслідків, аж до летальних. Своєчасна допомога відіграє важливу роль у подальшому лікуванні потерпілих і хворих, сприяє скороченню термінів їх медичної та трудової реабілітації [17].

Сторонні тіла можуть потрапляти у ніс за різних обставин. Частіше вони зустрічаються у дітей, які ховають у порожнині носа дрібні іграшки чи їх частини, металеві кульки, зерна бобових, злаків, кісточки вишень тощо. Іноді психічно хворі особи вводять собі в ніс сторонні тіла (як правило, відносно великих розмірів). Під час виробничих травм у порожнину носа може потрапити розплавлений метал, уламки скла, металу, каміння тощо. Харчові продукти стають сторонніми тілами під час блювання чи прийому їжі, особливо у хворих з парезом м’якого піднебіння. Живі сторонні тіла (п’явки, глисти) потрапляють у ніс при блюванні чи питті забрудненої води.

Сторонні тіла можуть себе нічим не проявляти. Частіше у хворих спостерігаються такі симптоми:

– Утруднене дихання, переважно однобічне.

– Слизисто-гнійні виділення з однієї половини носа, що мають неприємний запах і можуть мати домішки крові.

– Мацерація шкіри входу в ніс.

– Кровотеча (при сторонніх тілах з гострими краями).

– Головний біль.

– Прогресуючий кератит (запалення рогівки ока).

При тривалому перебуванні стороннього тіла у носі (роками) утворюються риноліти – нашарування на стороннє тіло солей вуглекислого і фосфорнокислого кальцію. Невеликі сторонні тіла іноді видаляють при сильному сяканні носа. Округлі сторонні тіла видаляють зігнутим гачком. Вклинені та сторонні тіла неправильної форми усувають пінцетом або носовими щипцями. Риноліти спочатку подрібнюють на фрагменти і видаляють щипцями частинами. Великі риноліти, які не вдається подрібнити, видаляють шляхом операції. Металеві сторонні тіла іноді видаляють електромагнітом.

Не можна видаляти кулясті тверді предмети з носа (кульки з підшипників, кісточки вишень тощо) пінцетом, бо це може призвести до зштовхування стороннього тіла вглиб і потрапляння його у нижні дихальні шляхи. Найбільш часта локалізація сторонніх тіл носа та глотки [18].

3.1.2 Сторонні тіла глотки

Сторонні тіла переважно потрапляють у глотку з порожнини рота під час їди, значно рідше – з порожнини носа, гортані чи стравоходу. Сторонні тіла, що залишаються у глотці, найчастіше заносяться сюди з їжею, наприклад, риб’ячі кістки, голки, шпильки тощо. Сприяють цьому раптовий кашель, сміх, чхання, розмова, які від-вертають увагу від акту жування. Потрапляння сторонніх тіл з порожнини рота полегшується при наявності зубних про-тезів, які погіршують контроль з боку твердого піднебіння. У стінки глотки можуть також проникати предмети, що тримають в роті діти під час гри, дорослі (шевці, кравці) під час роботи (цвяхи, кнопки, шпильки і ґудзики); уламки зубних протезів. Звичайно в

глотці зупиняються лише ті сторонні тіла, які встромились своїм гострим кінцем у слизову оболонку зіва, ділянку мигда-ликів чи піднебінних дужок. Більші й гладенькі сторонні тіла частіше проходять у гортаноглотку – до входу в стравохід та гортань, або зупиняються у са-мому стравоході. Рідше сторонні тіла локалізуються у носоглотці. Сюди можуть потрапити через ніс сторонні тіла, що відломилися в результаті травми, або рештки їжі при блюванні. З живих сторонніх тіл іноді сюди потрапляють п'явки – при питті болотної води, в носо-глотку можуть потрапляти аскариди.

Симптоми сторонніх тіл можуть бути різноманітними і залежать від їх характеру. Скарги хворих при гострих сторонніх тілах, що заглибилися в слизову оболонку, зводяться до колючого болю, який посилюється при ковтанні. Найчастіше сторонні тіла знаходяться в мигдалику, в піднебінних дужках та корені язика. При тривалому перебуванні стороннього тіла виникає запалення слизової оболонки глотки у вигляді гіперемії, набряку і підвищеної продукції слизу, що, у свою чергу, спричиняє кашель, слинотечу, іноді – блювання. При зупинці великого стороннього тіла в нижньому відділі глотки, біля входу в гортань, можуть розвинутись рап-тові явища ядухи. Іншим проявом сторонніх тіл нижніх відділів глотки або стравоходу є "озера" слини у грушоподібних синусах. Виявити гострі сторонні тіла, що застрягли у глотці (риб'я-ча кістка) можна при змащуванні слизової оболонки ватником на зонді з нарізкою. Шматок вати, що залишився на слизовій оболонці, зачепившись за кінчик стороннього тіла, вказує на його локалізацію у глотці.

Ретельним і системним огля-дом усіх відділів глотки вдається виявити стороннє тіло. Видаляють його за допомогою прямого або зігнутого корнцанга з браншами, що щільно стикаються між собою. Іноді подряпини чи садна, які залишилися після проковтування сто-роннього тіла, непокоять хворого і симулюють його перебування у глотці. У таких випадках рекомендують пов-торний огляд через 1-3 доби. Після видалення стороннього тіла призначають дезінфікуючі полоскання, щадну дієту. Передбачає уважне спостереження за пацієнтом з визначенням загального стану та вимірюванням температури,

призначення щадної дієти. Про будь-які прояви запальної реакції повідомляють лікаря, оскільки можливий розвиток септичних ускладнень чи стенозу гортані [18].

3.1.3 Сторонні тіла вуха

Сторонні тіла зовнішнього слухового проходу найчастіше зустрічаються у дітей віком 3-7 років. Діти заштовхують собі у вухо різноманітні предмети: дрібні іграшки чи їх частини, ґудзики, намистинки, насіння, горох, плодові кісточки тощо. У дорослих зустрічаються уламки сірників, жмутки вати, зерна злаків, гілки кущів, частини металу, вугілля тощо. Рідше трапляються живі сторонні тіла, переважно членистоногі: блощиці, таргани, кліщі, що заповзають у вухо. При хроніч-них середніх отитах, особливо коли виділення з вуха мають неприємний запах, у вусі можуть бути личинки мух.

Якщо сторонні тіла гладенькі й не травмують стінок слухового проходу, то вони можуть протягом довгого часу не викликати скарг у хворого. При тривалому перебуванні у вусі на стороннє тіло нашаровується сірка і його стає важко розпізнати при отоскопії. Більшу небезпеку для хворого становить не саме стороннє тіло, а невміла спроба його видалення! Предмети з гострими зазубреними краями можуть бути причиною неприємних відчуттів та зниження слуху. Живі сторонні тіла своїми рухами можуть викликати різкий біль у вусі та шум [17].

Перед тим як видалити стороннє тіло, треба обов’язково провести отоскопію та визначити його форму і характер. Найбезпечнішим способом видалення сторонніх тіл є вимивання, що проводиться так само, як і при видаленні сірчаної пробки. Якщо при дослідженні виявлено, що вимивання є недоцільним, то стороннє тіло можна видалити пінцетом або спеціальним тупим гачком, що повинен зробити лікар. Плоскі сторонні тіла видаляють вушним пінцетом або мікрощипцями. Округлі тверді сторонні тіла ніколи не слід видаляти пінцетом або щипцями, бо такими інструментами захопити їх не вдається, а при

спробі фіксувати такі тіла висковзують і ще глибше заштовхуються в зовнішній слуховий прохід. Такі сторонні тіла видаляють гачком, який заводять за тіло і витягують назовні, що повинен проводити лікар. Набрякаючі й защемлені у зовнішньому слуховому проході сторонні тіла, такі, як бобові (горох, квасоля), кукурудза тощо, піддають попередньому зневодненню шляхом повторного вливання у вухо концентрованого (96°) спирту і видаляють інструментами. Якщо видалити стороннє тіло не вдалось через різке запалення стінок зовнішнього слухового проходу чи глибоке його проникнення в кістковий відділ проходу, то його видаляють шляхом операції. Живі сторонні тіла видаляють так: спочатку закапують у вухо спирт або олію, щоб вбити або знерухомити комаху, а потім промивають шприцом Жане чи видаляють спеціальними інструментами.

Після вимивання неускладненого стороннього тіла слуховий прохід обробляють спиртом та прикривають кулькою стерильної вати. Вату видаляють через 30 хв, пацієнту рекомендують уникати потрапляння води у вухо протягом 5-7 діб. При ускладнених сторонніх тілах, залежно від наявності й характеру пошкоджень, лікування проводять так само, як при травмах зовнішнього вуха [18].

3.2 Особливості заходів електробезпеки на підприємствах.

Технічні засоби і заходи з електробезпеки реалізуються в конструкції електроустановок при їх розробці, виготовленні і монтажі відповідно до чинних нормативів. За своїми функціями технічні засоби і заходи забезпечення електробезпеки поділяються на дві групи:

- технічні заходи і засоби забезпечення електробезпеки при нормальному режимі роботи електроустановок;

- технічні заходи і засоби забезпечення електробезпеки при аварійних режимах роботи електроустановок.

Із метою підвищення рівня безпеки, залежно від призначення, умов експлуатації і конструкції, в електроустановках застосовується одночасно більшість з перерахованих технічних засобів і заходів. Ізоляція струмовідних частин. Забезпечує технічну працездат-ність електроустановок, зменшує вірогідність потраплянь людини під напругу, замикань на землю і на корпус електроустановок, зменшує струм через людину при доторканні до неізольованих струмовідних частин в електроустановках, що живляться від ізольованої від землі мережі за умови відсутності фаз із пошкодженою ізоляцією. Розміщення струмовідних частин на недосяжній висоті чи у недоступному місці стосується проводів повітряної лінії (ПЛ) чи шин та обладнання на підстанціях. Висота розміщення проводів залежить від напруги лінії та місцевості, якою вона проходить. Наприклад, висота розміщення проводів ПЛ напругою до 110 кВ включно у населеній місцевості повинна бути не менш як 7 м над місцевістю (землею). Розміщення струмовідних частин у недосяжному місці стосується кабельних ліній (КЛ) чи проводів у приміщенні. КЛ прокладають у землі у траншеях чи спорудах на глибині більш як 0,6 м. Проводку у приміщенні прокладають під шаром штукатурки або під підлогою.

Основні вимоги до будівель виробничого призначення викладені в СНиП 2.09.02-85. При плануванні виробничих приміщень необхідно враховувати санітарну характеристику виробничих процесів, дотримуватись норм корисної площі для працюючих, а також нормативів площ для розташування устаткування і необхідної ширини проходів, що забезпечують безпечну роботу та зручне обслуговування устаткування. Вибір типу приміщення визначається технологічним процесом та можливістю боротьби з шумом, вібрацією і забрудненням повітря. Наявність великих за розміром віконних прорізів та ліхтарів має забезпечувати хороше натуральне освітлення. Обов’язковим являється також улаштування ефективної вентиляції. Якщо в одній будові необхідно розмістити виробничі приміщення, до яких з точки зору промислової санітарії та пожежної профілактики висуваються різні вимоги, то необхідно їх групувати таким чином, щоб вони були ізольованими один від одного. Цехи,

відділення та дільниці зі значними шкідливими виділеннями, надлишком тепла та пожежонебезпечні необхідно розташовувати біля зовнішніх стін будівлі і, якщо допустимо за умовами технологічного процесу та потоковістю виробництва — на верхніх поверхах багатоповерхової будівлі. Не можна розташовувати нешкідливі цехи та дільниці (наприклад, механоскладальні, інструментальні, ЕОМ тощо), а також конторські приміщення над шкідливими, оскільки при відкриванні вікон гази та пари можуть проникати в ці приміщення. Приміщення, де розташовані електрощитове, вентиляційне, компресорне та інші види обладнання підвищеної небезпеки повинні бути постійно зачиненими на ключ, з тим, щоб в них не потрапили сторонні працівники. З метою запобігання травматизму у виробничих приміщеннях необхідно застосовувати попереджувальне пофарбування будівельних конструкцій та знаки безпеки (ГОСТ 12.4.026-76 „Цвета сигнальные и знаки безопасности"). Наприклад, жовтим кольором (або із чорними смугами) фарбують низько розташовані над проходами конструкції, звуження проїздів, малопомітні сходинки, виступи та перепади в площині підлоги. Ширина основних проходів всередині цехів та дільниць повинна бути не менше 1,5 м, а ширина проїздів — 2,5 м. Двері та ворота, що ведуть безпосередньо на двір, необхідно обладнати тамбурами або повітряними (тепловими) завісами. Порядок розташування устаткування і відстань між машинами визначаються їхніми розмірами, технологічними вимогами і вимогами техніки безпеки. Однак, у всіх випадках, до устаткування, що має електропривід, повинен бути вільний підхід з усіх сторін шириною не менше 1 м зі сторони робочої зони і 0,6 м — зі сторони неробочої зони. Виробничі меблі (шафи, стелажі, столи тощо) можна ставити впритул до конструктивних елементів будівлі — стін, колон. Для обробки та захисту внутрішніх поверхонь конструкцій приміщень від дії шкідливих та агресивних речовин (наприклад, кислот, лугів, свинцю) та вологи використовують керамічну плитку, кислотостійку штукатурку, олійну фарбу, які перешкоджають сорбції цих речовин та допускають миття поверхонь. Висота виробничих приміщень повинна бути не менше 3,2 м, а об’єм і площа — 15 $м^3$ та 4,5 $м^2$ відповідно на кожного працівника. Стіни виробничих та побутових приміщень

мають відповідати вимогам шумо- і теплозахисту; підлягати легкому прибиранню та миттю; мати оздоблення, що виключає можливість поглинення чи осадження отруйних речовин (керамічна плитка, олійна фарба). Приміщення, де розміщені виробництва з віділенням шкідливих та агресивних речовин (кислоти, луги, ртуть, бензол, сполуки свинцю та ін.), повинні мати стіни, стелю та конструкції, пофарбовані так, щоб попереджувалась сорбція (осаджування) цих речовин та допускались легкі очищення та миття цих поверхонь. У приміщенях з великим виділенням пилу (шліфування, заточка, розмел) слід передбачити прибирання за допомогою пилососів чи гідрозмивання. Колір інтер'єрів приміщень має відповідати вимогам технічної естетики [18].

ВИСНОВОК

Розроблено медичну інформаційну систему. Можливості даної системи дозволяють відображати дані про пацієнта не тільки у вигляді традиційних розділів медичної карти, а й представляти інформацію в зрізах, що цікавлять лікаря-невролога. Доступ до бази даних відбувається через інтерфейс клієнтського програмного застосування лікаря-невролога.

Метою даної роботи було створення автоматизованого робочого місця лікаря-невролога. Необхідно оптимізувати робочий процес з метою економії часу співробітника. Таким чином, у рамках цієї роботи була вивчена та застосована одна з найпопулярніших методологій гнучкої розробки Scrum. У наступному пункті створення програми було складено моделі AS-IS ключового бізнес-процесу за допомогою нотації UML Activity Diagram. Далі для реалізації ПЗ були ідентифіковані способи аналізу вимог та виявлені користувацькі та функціональні вимоги, а також створена матриця відстеження вимог. Ще в цій таблиці було виявлено пріоритет за допомогою методу MuSCoW. Після цього було складено план створення програмного забезпечення, згідно з методологією SCRUM. Усього вийшло 4 спринти.

Розробка програми проводилася серед створення та обробки баз даних MS Access. Ця програма є простою та зрозумілою у зверненні; має можливість внесення даних із програми Microsoft Excel, де раніше велася частина баз даних; є повністю сумісною з системою Windows і має величезні можливості з імпорту та експорту даних. За підсумками були в MS Access були реалізовані вимоги, виявлені недоліки та виправлені помилки [18].

СПИСОК ЛІТЕРАТУРИ

1. Джеф Сазерленд. Scrum. Революційний метод керування проектами Scrum. Зображення робить twice роботу в першому часі. – Манн, Іванов та Фербер, 2016. – 288 с.

2. Орел А.А., Ромакіна О.М. Навчальний посібник з курсу «Проектування бізнес-процесів» для студентів механіко-математичного факультету, 2007. – 123 с.

3. А.В. Варзунов, Є.К. Торосян, Л.П. Сажніва. Аналіз та управління бізнес-процесами. Навчальний посібник, 2010 – 212 с.

4. Скоромець А.А., Скоромець А.П., Скоромець Т.А., Неврологічний статус та його інтерпретація. Оформлення, оригінал-макет. Видавництво "МЕДпрес-інформ", 2009. - 439 с.

5. Г.М. Жданів. Схема історії хвороби неврологічного хворого, 2008. - 128 с.

6. Лайза Кріспін, Джанет Грегорі. Гнучке тестування: практичний посібник для тестувальників програмного забезпечення та гнучких команд Agile Testing: A Practical Guide for Testers and Agile Teams. - К, 2010. - 464 с.

7. Назаренко Г. І., Гулієв Я. І., Єрмаков Д. Є. Медичні інформаційні системи: теорія та практика. - К, 2015. - 320 с.

8. Шеєр А.В. Моделювання бізнес-процесів. – ЇХ., 2000. - 224 с.

9. Буч Р, Рамбо Дж. Введення в UML від творців мови. – 2015. – 493 с.

10. Карл І. Вігерс. Розробка вимог до програмного забезпечення. - 2005. - 576 с.

11. Соммервілл І. Інженерія програмного забезпечення. 2005. - 624 с.

12. Вендер А.М. CASE технології. Сучасні методи та засоби проектування інформаційних систем М.: Фінанси та статистика, 2005. - 352 с.

13. Ульман, Дж. Основи систем баз даних/Дж. Ульман. -, 2017. - 292 с.

14. Єпанешников А.М., Єпанешников В.А. Практика створення програм у Access. - 2009. - 440 с.

15. Бекаревич Ю., Пушкіна Н. Самовчитель MS Office Access 2016. - 464 с.

16. Федорів І.Г. Системний підхід до виявлення бізнес-процесів методом «згори донизу» // Прикладна інформатика. - 2012. № 5 (41), - С.5-13.

17. Криницький, Н.А. Автоматизовані інформаційні системи/Н.А. Криницький, Г.А. Миронов, Г.Д. Фролів. - К.: Наука, 2016. - 382 с.

18. Мягченко О. Безпека життєдіяльності людини та суспільства. – 2017. - 384 с.

19. Хаббард, Дж. Автоматизоване проектування баз даних/Дж. Хаббард. - 2014. - 296 с.

20. Ларман Крег. Застосування UML та шаблонів проектування. - 2015. - 624 с.

21. Мюллер, Р.Дж. Бази даних та UML. Проектування/Р.Дж. Мюллер. - 2017. - 420 с.

22. Бази даних (MS Access, MySQL) / А.А. Аббакумов, В.Л. Акімов, А.І. Єгунова, К.А. Лещанкін, В.М. Таланов. - 2011. - 112 с.

23. Методичні вказівки до виконання дипломної роботи освітнього рівня ―бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення‖ / Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

24. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.

25. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

26. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Моделювання та видобуток даних (висопродуктивні обчислення у великих алгебраїчних та числових системах, комбінаторному аналізі): навчальний посібник. Тернопіль: : ТНТУ 2019 – 62 с.

27. Нога В.В., Цуприк Г.Б. Інформаційні технології при створенні автоматизованих систем для задач збору та збереження інформації. Збірник тез доповідей VII Міжнародної науково-технічної конференції молодих учених та студентів. Актуальні задачі сучасних технологій – Тернопіль 28-29 листопада 2018. — Т. : ТНТУ, 2018. — Том 2. — С. 131.

28. І.В. Бойко, М.Р. Петрик, Г.Б. Цуприк. Дискретні структури (Алгебраїчні та числові системи, комбінаторний аналіз). Навчально-методичний посібник для студентів спеціальності 121 «Інженерія програмного забезпечення», аспірантів та викладачів вищих навчальних закладів. Конспект лекцій : Навчальний посібник / І.В. Бойко, М.Р. Петрик, Г.Б. Цуприк – Тернопіль : ТНТУ , 2017. – 64 с.

ДОДАТКИ

ДОДАТОК А

Тест план та результати тестування

На лістингах 2.1 та 2.2 редставлено реалізацію модульних тестів методів створення та редагування даних відповідно.

клас Cwork

```
# include
# include
# include "Worker. h "CWork
{::: vector pr; WorkBegin;
WorkEnd;
Work [20];
WorkPlace [25];
Reason [40];:
getWorkBegin (void);
setWorkBegin (int AWorkBegin );
getWorkEnd (void); setWorkEnd (int AWorkEnd);
* getWork (void);
setWork (char * AWork);
* getWorkPlace (void);
setWorkPlace (char * APlace);
* getReason (void);
setReason (char * AReason );
AddWorkerIntoWork (CWorker * g);
RemoveWorker (CWorker * g);
CountWorkerInWork ();
WriteV ();
WriteFV (std :: ofstream & t);
ReadFV (std :: ifstream & t);
Read ();
Write ();
ReadF (std :: ifstream & t);
WriteF (std :: ofstream & t);
();
(CWork & As);
(int AWorkBegin, int AWorkEnd, char * AWork, char * AWorkPlace , char * AReason);
};
```

клас Cworker

```
# ifndef _PEO__
# define _PEO__
# include
# include CWorker
{: PersonID; * Family; Birth [12];: getPersonID (void); setPersonID (int APersonID); * getFamily
(void); setFamily (char * AFamily); * getBirth (void) ; setBirth (char * ABirth); Read (); Write ();
ReadF (std :: ifstream & f); WriteF (std :: ofstream & f); (); (CWorker & As); (int APersonID, char *
AFamily, char * ABirth);
~ CWorker ();
};
# endif
```

клас CDocent

```
# include
```

```
# include "Worker. h "CDocent: public CWorker
{: GodZ; UchSpec [15];: getGodZ (void); setGodZ (int AGodZ); * getUchSpec (void); setUchSpec
(char * AUchSpec); Write (); Read (); WriteF (std :: ofstream & t); ReadF (std :: ifstream & t); ();
(CDocent & As); (int APersonID, char * AFamily, char * ABirth, int AGodZ, char * AUchSpec );
};
```

Файл Work. cpp

```
# include "stdafx. h "
# include
# include "Work. h "
# include "Worker. h "
# include "Docent. h "
# include
# include std :: vector v; std :: vector w; std :: vector v2; CWork :: AddWorkerIntoWork (CWorker *
g)
{. push_back (g);
gn="justify">} CWork :: WriteV ()
{<<"Kollichestvo rabotnikov:" <
{[i] -> Write ();
}
} CWork :: WriteFV (std :: ofstream & t)
{<
{(int n = 0; n
{(v [n] == pr [i]) break;
} <
}
} CWork :: ReadFV (std :: ifstream & t)
{N2, n2;>> N2; (int i = 0; i
{>> n2;. push_back (v [n2]);
}
} CWork :: CountWorkerInWork ()
{pr. size ();
} CWork :: getWorkBegin (void) {WorkBegin;
} CWork :: setWorkBegin (int AWorkBegin) {= AWorkBegin;
} CWork :: getWorkEnd (void) {WorkEnd;
} CWork :: setWorkEnd (int AWorkEnd) {= AWorkEnd;
} * CWork :: getWork (void) {Work;
} CWork :: setWork (char * AWork) {(Work, AWork);
} * CWork :: getWorkPlace (void) {WorkPlace;
} CWork :: setWorkPlace (char * AWorkPlace) {(WorkPlace, AWorkPlace);
} * CWork :: getReason (void) {Reason;
} CWork :: setReason (char * AReason) {(Reason, AReason);
} CWork :: Read ()
{<<"Vvedite WorkBegin" <> WorkBegin; <<"Vvedite WorkEnd" <> WorkEnd ; <<"Vvedite Work"
<> Work; <<"Vvedite WorkPlace" <> WorkPlace; <<"Vvedite Reason "<> Reason; i; command = 1; *
Temp100; (command! = 0)
{<<"Please enter operation." <> command; (command == 1)
{= new CWorker (); -> Read ();. push_back (Temp100); (Temp100);
} if (command == 2)
{<<"Vvedite nomer Workera iz globalnogo spiska";>> i; (v [i]);
}
{<<"Invalid command" <
}
```

```
}
} CWork :: Write ()
{<<"WorkBegin -" <
} CWork :: ReadF (std :: ifstream & t)
{>> WorkBegin>> WorkEnd>> Work>> WorkPlace>> Reason; (t);
} CWork :: WriteF (std :: ofstream & t)
{<
<
} :: CWork ()
{= 1990; = 2000; (Work, "Director"); (WorkPlace, "Lotus");
strcpy (Reason, "Po sobstvennomu jelaniyu");
} :: CWork (CWork & As)
{= As. WorkBegin; = As. WorkEnd; (Work, As. Work); (WorkPlace, As. WorkPlace); (Reason, As.
Reason);
} :: CWork (int AWorkBegin, int AWorkEnd, char * AWork, char * AWorkPlace, char * AReason)
{= AWorkBegin; = AWorkEnd; (Work, AWork); (WorkPlace, AWorkPlace); (Reason, AReason);
}
```

Файл Worker. cpp

```
# include "StdAfx. h "
# include
# include "Worker. h "
# include CWorker :: getPersonID (void) {PersonID;
} CWorker :: setPersonID (int APersonID) {= APersonID;
} * CWorker :: getFamily (void) {Family;
} * CWorker :: getBirth (void) {Birth;
} CWorker :: setBirth (char * ABirth)
{(Birth, ABirth);
} CWorker :: Read ()
{<<"Vvedite PersonID" <> PersonID; <<"Vvedite Family" <> Family ; <<"Vvedite Birth" <> Birth;
} CWorker :: Write ()
{<<"PersonID -" <
} CWorker :: ReadF (std :: ifstream & f)
{>> PersonID>> Family>> Birth;
} CWorker :: WriteF (std :: ofstream & f)
{<
} :: CWorker ()
{= 10; = new char [14]; (Family, "Ivanov"); (Birth, "01.01.10");
} :: CWorker (CWorker & As)
{= As. PersonID; = new char [strlen (As. Family) +1]; (Birth, As. Birth);
} :: CWorker (int APersonID, char * AFamily, char * ABirth)
{= APersonID; = new char [strlen (AFamily) +1]; (Family, AFamily); (Birth, ABirth);
} :: ~ CWorker ()
{[] Family;
} CWorker :: setFamily (char * AFamily)
{(Family! = 0) [] Family; = new char [strlen (AFamily) +1]; (Family, AFamily);
}
```

Файл Docent. cpp

```
# include "stdafx. h "
# include
# include "Docent. h "
# include "Worker. h "
>
```

```
# include CDocent :: getGodZ (void)
{GodZ;
} CDocent :: setGodZ (int AGodZ)
{= AGodZ;
} * CDocent :: getUchSpec (void) {UchSpec;
} CDocent :: setUchSpec (char * AUchSpec) {(UchSpec, AUchSpec);
} :: CDocent (): CWorker ()
{= 1997; (UchSpec, "Docent");
} :: CDocent (int APersonID, char * AFamily, char * ABirth, int AGodZ, char * AUchSpec): CWorker
(APersonID, AFamily, ABirth)
{= AGodZ; (UchSpec, AUchSpec);
} :: CDocent (CDocent & As)
{= As. GodZ; (UchSpec, As. UchSpec);
} CDocent :: Read (void)
{<<"vvedite informazhiy o GodZ and UchSpec" <> GodZ; <<"vvedite UchSpec" <> UchSpec;
} CDocent :: ReadF (std :: ifstream & t)
{:: ReadF (t);>> GodZ>> UchSpec;
} CDocent :: Write (void)
{<
} CDocent :: WriteF (std :: ofstream & t)
{:: WriteF (t); <
}
```

Файл main. cpp

```
# include "stdafx. h "
# include
# include
# include
# include "Work. h "
# include "Worker. h "
# include "Docent. h "
# include ReadFAllCWorker (char * tn, std :: vector & v)
{:: ifstream t (tn); N;>> N; * m; (int i = 0; i
{= new CWorker ();> ReadF (t);. push_back (m);
}. close ();
} WriteFAllCWorker (char * tn, std :: vector v)
{:: ofstream t (tn); <
{[i] -> WriteF (t);
}. close ();
} WriteAllCWorker (std :: vector v)
{<<"kolich Worker =" <
{[i] -> Write ();
}
} ReadFAllCWork (char * tn, std :: vector & w)
{:: ifstream t (tn); N;>> N; * m; (int i = 0; i
{= new CWork ();> ReadF (t);. push_back (m);
}. close ();
} WriteFAllCWork (char * ...
tn, std :: vector w)
{:: ofstream t (tn); <
{[i] -> WriteF (t);
}. close ();
} WriteAllCWork (std :: vector w)
```

```
{<<"kolich Workov =" <
{[i] -> Write ();
}
} ReadFAllCDocent (char * tn, std :: vector & v2)
{:: ifstream t (tn); N;>> N; * m; (int i = 0; i
{= new CDocent ();> ReadF (t);. push_back (m);
}. close ();
} WriteFAllCDocent (char * tn, std :: vector v2)
{:: ofstream t (tn); <
{[i] -> WriteF (t);
}. close ();
} WriteAllCDocent (std :: vector v2)
{<<"kolich specialnostei =" <
{[i] -> Write ();
}
} :: vector v; :: vector w; :: vector v2; main (void)
{("Worker. txt", v); ("Work. txt", w); ("Docent. txt", v2); tn [600]; kod; { < vector "< File "< Display
"< vector "< File "< Display "< vector "< File "< Display "<> kod; (kod)
{1: <> tn; (tn, v);; 2: <> tn; (tn, v);; 3: (v);; 4: * m; = new CWorker;> Read ();. push_back (m);; 5: <>
tn; (tn, w);; 6: << ; endl; <<"Input File name:" <> tn; (tn, w);; 7: (w);; 8: * a; = new CWork; > Read
();. push_back (a);; 9: <> tn; (tn, v2);; 10: << ; endl; <<"Input File name:" <> tn; (tn, v2);; 11: (v2);;
12: * o; = new CDocent; > Read ();. push_back (o);;
}
} while (kod> 0); <<"GOOD BYE";
}
```

ДОДАТОК В