МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему:

«Інформаційна система автоматизації виробництва бурильних труб»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Ободяк В.К.**

Студента групи ІН.дн – 61с — **Кучерявенко Б.М.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет ЦЗДВФН Кафедра Комп'ютерних наук
Спеціальність «Комп'ютерні науки»

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Кучерявенку Богдану Михайловичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Інформаційна система автоматизації виробництва бурильних труб

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____
3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз існуючих рішень. Постановка задачі дослідження. 2) Вибір інструментів для створення інформаційної системи. 3) Програмна розробка інформаційної системи обліку.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Аналіз існуючих рішень. Постановка задачі дослідження		
2.	Модель інформаційної системи автоматизації виробництва бурильних труб		
3.	Програмна розробка інформаційної системи обліку		
4.	Оформлення пояснювальної записки до дипломної роботи		

Студент _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 65 стор., 22 рис., 1табл., 1 додаток, 16 джерел.

Об'єкт дослідження — процес роботи інформаційної системи автоматизації виробництва бурильних труб на ДП «Завод ОБ та ВТ».

Мета роботи — розробити інформаційну систему автоматизації виробництва бурильних труб. В цьому проекті повинно бути розроблено систему автоматизації виробництва бурильних труб що повністю відповідає завданням які сформував замовник. Програмний продукт призначений для автоматизації процесів виробництва, контролю якості, розрахунків собівартості та кінцевої вартості виробів а також дає можливість відстежувати в якій фазі в будь-який момент находиться виконання замовлення покупця. Інтерфейс дозволяє користувачу працювати з документами вигляд яких повністю відповідний до паперових документів, що були затверджені до використання на підприємстві. Продукт дозволить використовувати вже наявні програмні рішення як то MS SQL server 2017 standard та MS Windows Server 2019 standard, що дозволить збудувати швидку, стійку та безпечну в використанні систему.

Методи дослідження — метод проектування програмного забезпечення та перевірки працездатності системи в реальному часі.

Результати — було розроблено інформаційну систему, здійснено дослідно-промислову експлуатацію, продукт рекомендовано до використання.

ЗМІСТ

ВСТУП

Тема бакалаврської роботи – інформаційна система автоматизації виробництва бурильних труб, яка впроваджується на ДП «Завод ОБ та ВТ».

Технічним завданням розробки системи є забезпечення повного контролю, оптимізації всього циклу виробництва бурильних труб на підприємстві починаючі від формування закупівель сировини і до отримання клієнтом відповідної продукції. Система має бути сумісною з сучасним комп'ютерним обладнанням та існуючим на виробництві обладнанням, яке відповідає застарілому стандарту Siemens Simatic S5.

Також система має у собі довідкову частину, яка на приклад допоможе працівнику в нетипових ситуаціях обрати вірне рішення стосовно корективи виробничого процесу.

Окремо варто зазначити складову безпеки. За даними infowatch [8] щорічно кількість витоків конфіденційних даних зростає щонайменше на 16%. Тому вихідний код додатку обов'язково повинен бути перевірений на відсутність недокументованих функцій. Для того, щоб бути успішним та ефективним підхід до процесу автоматизації має бути зваженим та послідовним. Необхідно чітко будувати логіку та враховувати комплексні завдання. Як що поставитися до завдання автоматизації як до «латання дір» це може призвести до неможливості гнучкої адаптації системи в майбутньому. Треба відзначити необхідність чітко стандартизувати усі процеси на підприємстві, як приклад на ДП «Завод ОБ та ВТ» це зроблено за допомогою «Стандартів підприємства» скорочено СТП. Кожна виробнича, дослідницька або організаційна дія вимагає часу і як наслідок коштує грошей, саме стандартизація дозволяє уникнути потреби переробляти ці дії як що вони стандартизовані. У випадку коли це є економічно обґрунтоване, гарним рішенням є повна модернізація обладнання підприємства. Але й вона не є панацеєю, бо потребує ще й перепідготовку персоналу. Також і сам процес виробництва є не окремим використанням технологій, але комплекс рішень,

що складаються з фінансових показників, планування, логістики, пов'язаних в єдину систему, єдиний організм.

Автоматизація підприємства це перший та найефективніший етап для оптимізації витрат на підприємстві. Для повноцінного запровадження автоматизації на підприємстві потрібно дуже детально та влучно розроблене ТЗ. Наприклад, маємо змогу планувати заздалегідь замовлення транспорту для доставки товарів покупцю, при цьому мінімізуючи час простою. Або Оптимізувати використання електроенергії просто плануючи виробничі операції на потрібну тарифну зону. За допомогою статистики використання витратних матеріалів, можна коригувати складські об’єми та схеми розміщення товарів. Формування простих для сприйняття таблиць та графіків скорочує час ухвалення рішень відповідальними особами.

Актуальність теми.

З огляду на те, що спеціалізованих додатків які повністю враховують специфіку підприємства немає, на цей момент найбільш поширеним додатком, який дає можливість керувати процесом виробництва є 1С [11].

Але є одне деякі зауваження:

1. Додаток 1С складно оптимізувати під конкретне промислове виробництво.
2. Виробником додатку є російська федерація, яка не є дружньою до нас.

Розробка додатку під окреме підприємство складна та доволі коштовна робота, але вона гарантує повну відповідність виробничому процесу та відмінні результати.

Об‘єктом дослідження є процес роботи інформаційної системи автоматизації виробництва бурильних труб.

1 АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ

1.1 Огляд існуючих рішень.

Зараз у світі панує інформаційна ера. Інформація є найпотужнішим інструментом для оптимізації у будь-якій сфері діяльності людини. Натан Ротшильд сказав: «Хто володіє інформацією - той володіє світом». Тому аналіз та ефективне використання інформації надає вагомі переваги для бізнес процесів.

Традиційно впровадження системи управління підприємством вимагає значного часу. У великих корпораціях цей процес може зайняти кілька років за участю сотень експертів. Це не підходить для малих і середніх підприємств, тому багато постачальників просувають універсальне рішення з можливістю подальшої адаптації для відповідності потребам вашої організації. Така заготовка повинна зменшити час, необхідний на впровадження, і запобігти розповзання меж проекту, яке призводить до збільшення вартості багатьох проектів.

Залежно від потреб вашої організації, а також від бюджету, можливо ви забажаєте для початку придбати основні модулі, а потім, з плином часу, додавати інші. Як правило, основний модуль являє собою систему показників обліку, а також модулі, характерні для даної індустрії: управління інвентаризацією для дистриб'юторів, торговельні точки для компаній, що займаються роздрібною торгівлею, планування виробничих ресурсів для виробників і розрахунок вартості проектів для компаній, що надають професійні послуги. Постачальники, які говорять, що цей метод має переваги в порівнянні з інтегрованими системами, мають рацію, однак вам буде потрібно навчитися використовувати систему найкращим чином, що також потребує часу. Рішення має бути модульним, щоб ви могли мати можливість розширити його з часом, і в той же час повністю інтегрованим, щоб усі модулі могли з'єднуватися в режимі реального часу. Таким чином, ви можете бути упевнені, що система, принаймні, швидше доставлятиме інформацію і

координуватиме процеси ведення бізнесу. Вона також повинна обслуговувати процеси ведення бізнесу, не заважаючи реалізації початкових функцій. Розробники просуватимуть впровадження своїх напрацювань, проте немає сенсу переймати модель ведення бізнесу, властиву вашим конкурентам, якщо ви нині випереджаєте їх, ведучи торгівлю більше інноваційними методами. Зокрема будьте дуже обережні з розробниками, що бажають продати вам "промисловий шаблон" свого програмного забезпечення. На перший погляд цінність цих шаблонів дуже велика, оскільки їх пропонують як рішення, що дозволяє скоротити витрати і спростити процес впровадження, а також зменшити ризики для вашого бізнесу.

Залучення до обробки інформації сучасних апаратних та програмних засобів значно розширює горизонт можливостей.

Використання єдиного додатку для керування виробництвом робить координацію спільних дій між підрозділами легкою та результативною. Накопичування даних про нетипові випадки, аварійні ситуації та шляхи їх усунення або нормалізації з часом дає можливість не відволікати вузьких спеціалістів для втручання в виробничий процес.

Централізовані обробка та зберігання технічної інформації тобто креслення, технологічні картки, хімічний склад, вміст сировини та енергоресурсів, інструкції, що до налагодження обладнання, вирішує проблему нестачі кваліфікованого персоналу та зменшує час прийняття рішень на місцях. Коректні підрахунки витрат на виробництво зменшують собівартість кінцевих виробів, що збільшує прибуток підприємства. Можливість пошуку схожих продуктів або ситуацій також не тільки надає можливість прогнозувати розвиток подій, але й аналізувати ефективність прийнятих рішень.

Технологічний прогрес з часом загострює конкуренцію на виробничому ринку. Для підвищення конкурентоспроможності підприємства вимушені або підвищувати якість, або зменшувати вартість виробів, або обидва кроки одночасно. Оптимізація витрат є одним з найважливіших напрямків у будь

якій бізнес-діяльності. Автоматизація виробництва дозволяє аналізувати ефективність багатьох аспектів діяльності підприємства та при необхідності запроваджувати нові технології та методи. Починаючи з структуризації конструкторсько-технологічної документації, аналізу систем постачання та зберігання, виробничого процесу оптимізуються витрати часу та коштів, що дає змогу підприємству бути більш прибутковим. Будь яка сучасна економіка базується на принципі "гроші - товар - гроші", тому максимально можлива мінімізація втрат грошей на усіх етапах є пріоритетною для підприємства.

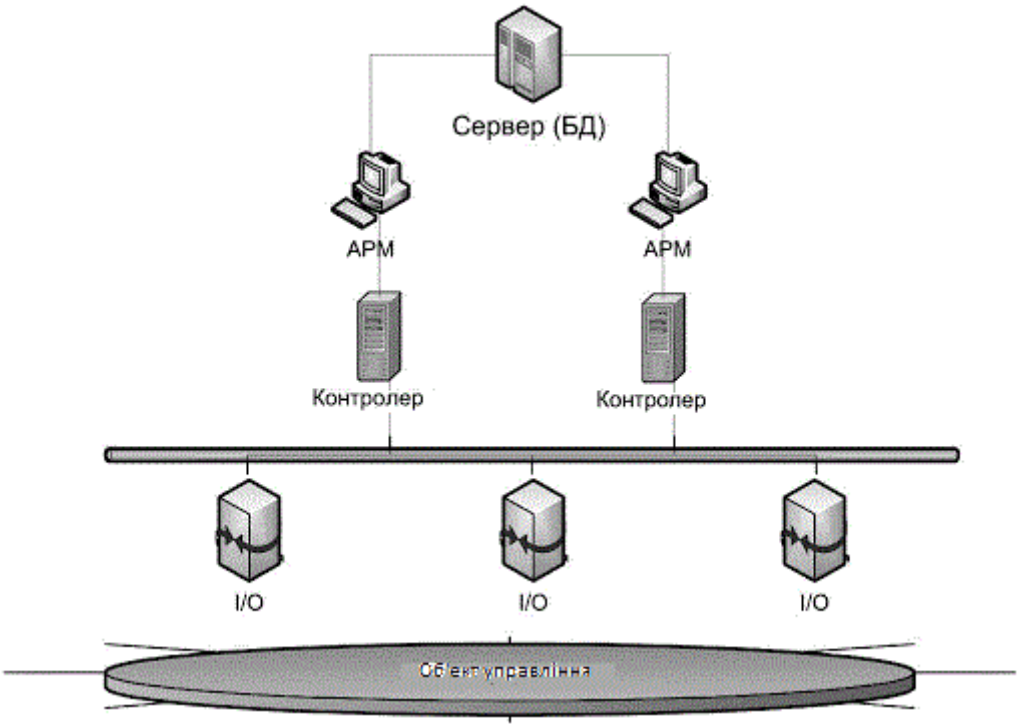


Рисунок 1.1 - Приклад структури SCADA системи

Розглянемо сучасне підприємство, яке використовує системи автоматизованого керування на рівнях: технологічного процесу (SCADA рис. 1.1), оперативного управління виробництвом (MES), автоматизації процесів проектування (CAD рис. 1.2), та системи ERP (керування діяльністю підприємства).

Рисунок 1.2 Структура CAD систем.

Коли мова йде про технології керування виробництвом, найчастіше, йдеться про MES рішення. Вони мають змогу обробляти як єдину базу економічні та технологічні процеси. Така інформаційна система допомагає вирішувати багато завдань важливих для виробничого підприємства, але не є достатньою. Найбільш ефективно використати усі засоби автоматизації наявні на підприємстві можливо завдяки створенню гетерогенної інформаційної системи, що є поєднанням різних рівнів керування. Наприклад планування матеріально-технічного забезпечення значно впливає на роботу складського господарства. Оскільки в реальному підприємстві немає нескінчених ресурсів будь-чого, невірне планування одного напрямку природньо викликає проблеми використання усіх пов'язаних напрямків або процесів. Тож для ефективного керування виробництвом потрібен єдиний інформаційний простір. Більшість виробничих підприємств у всьому світі, щодня стикаються

з різноманітними завданнями, аналіз і ефективне рішення яких, забезпечує стійке і конкурентоздатне положення підприємства на ринку. Одні пов'язані з територіальною розподіленністю виробничих комплексів і глобалізацією виробництва, що посилюється. Інші є наслідком неузгодженості внутрішніх процесів. Як наслідок відсутність чітких відпрацьованих процедур і ефективної взаємодії між підрозділами, відсутність об'єктивної інформації про найважливіші показники діяльності, що безпосередньо відбивається на конкурентоспроможності за такими показниками, як собівартість, якість і актуальність продукції, що випускається. Треті витікають з необхідності координувати власні бізнес-процеси із зовнішнім світом - вашими клієнтами, постачальниками, партнерами, і дистриб'юторами. Дуже часто через відсутність об'єктивної картини діяльності, яка ґрунтується на прозорих фінансових і виробничих показниках, складно реагувати вимоги ринку, що на постійно змінюються, і кон'юнктуру, що складається. Таким чином, ключовими орієнтирами в роботі залишаються збір і обмін інформацією, готовність до співпраці, гнучкість, мобільність і підтримка інновацій.

Бізнес-процеси для промислового сектора надзвичайно різноманітні: виробничі процеси, процеси матеріального забезпечення, процеси реалізації продукції, фінансові процеси, процеси сервісного обслуговування виробництва, процеси планування і управління ресурсами (забезпечення виробничими ресурсами), процеси взаємодії учасників колективного ухвалення рішень, процеси конструкторських і технологічних розробок.

Вимоги ринку, ринкова кон'юктура постійно змінюються. Для змоги адекватно реагувати на ці зміни підприємство повинно мати прозору виробничу та економічну модель інформації, змогу швидко запроваджувати інновації. Частіше за все бізнес-партнери надають перевагу гнучким та мобільним контрагентам. Завдяки прозорому і швидкому збору та обміну інформацією з'являється можливість досягти потрібних бізнес-якостей. Різномаїття бізнес-процесів на підприємстві є надзвичайним.

Це процеси матеріально-технічного забезпечення, планування та керування ресурсами, конструкторсько-технологічна діяльність, наради для колективного ухвалення рішень, стандартизація. На відміну від випадків планової економіки, підприємство ДП "Завод ОБ та ВТ" працює під замовлення, де параметри виробів надає покупець. Тож для успішної обробки замовлення необхідно сформувати привабливу цінову пропозицію для клієнта. Щоб сформувати ціну необхідно планувати ресурси необхідні для виробництва замовлення, ціну транспортування до замовника (як що це передбачено контрактом на постачання Згідно зі ст. 712 Цивільного кодексу), виробничі графіки. При безпосередньому виконанні замовлення автоматизована система надає допомогу у диспетчеризації виробництва, безпосередньому керуванні виробничими майданчиками, структурованому зберіганні готової продукції. Щоб забезпечити ефективність інформаційної системи необхідно використовувати кастомні (англ. custom, тобто розроблені під конкретне підприємство) конфігурації. Особливістю такої моделі виробництва є необхідність швидкого реагування на потік клієнтських замовлень, що змінюється, і формування оптимального виробничого процесу з метою максимального забезпечення рівня обслуговування замовників.

Що до виробництв які входять до великих концернів або мають велико серійні закази, відсутність швидкої зміни номенклатури заказів не спрощує процес керування. В цьому випадку все одно потрібно робити аналіз ринку та прогнозувати збут готової продукції. Всі інші аспекти виробництва співпадають з виробництвом під замовлення.

Однак при розробці під замовлення клієнт часто замовляє виріб який ще не вироблявся на даному підприємстві та під нього немає розробленої технічної та/або конструкторсько-технологічної документації. Так це вже вимагає розробки комплекту документації або запровадження нових технологій. Також присутні всі процеси які є при виробництві під замовлення. Конфігурація під замовлення це майже те саме що й виробництво під замовлення, але виробник пропонує клієнту вибір опцій (наприклад, зміни

розмірів у рамках доступних, зміна кольору) що не потребують розробки нової технічної та/або технологічної документації. При будь-якому різновиді виробництва дуже важливими є логістика та закупівлі (за для виконання плану постачання або під конкретне замовлення). Це також вимагає затрат на керування, наприклад розподіл навантаження між складськими комплексами.

Під автоматизацією підприємства можна розуміти різні аспекти. Необхідно розрізняти автоматизацію виробничих процесів ("Автоматизована система управління технологічним процесом" або АСУ ТП) і Повну автоматизацію виробництва (раніше - "АСУ-виробництво" або АСУП, нині - виробничі модулі ERP- системи [16].

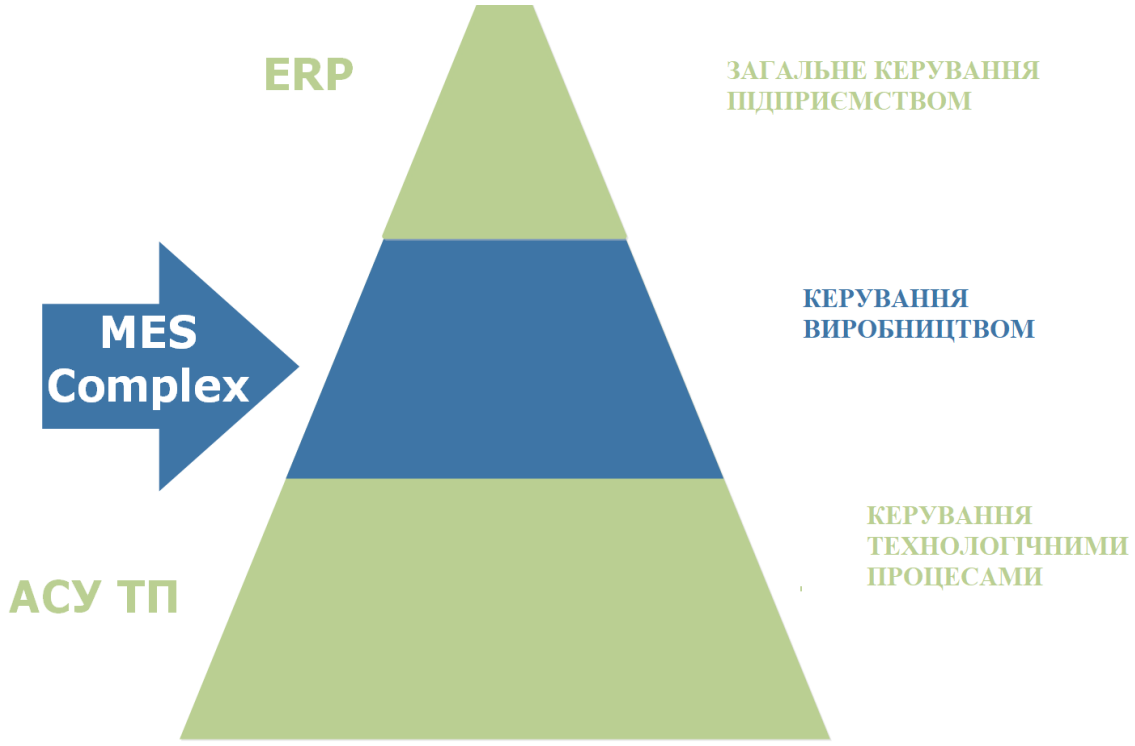


Рисунок 1.3 Ієрархія систем керування.

Автоматизована система управління технологічним процесом (АСУ ТП) це комплексна програмно-апаратна система, що розробляється керування технологічними процесами та технологічним обладнанням на виробничих підприємствах. Зазвичай системою АСУ ТП називають систему що забезпечує керування технологічними процесами на підприємстві в цілому, або керує циклом виробництва окремого продукту чи переліку продуктів. До складу

системи АСУ ТП можуть входити системи автоматичного управління (САУ), пристрої що мають засоби автоматизації та об'єднані в комплекси (наприклад комплекс станків з ЧПУ, що керується з одного пульту або одним оператором), системи SCADA що призначені для диспетчеризації та збору інформації також сюди можуть відноситися системи на програмованих логічних контролерах та розподілені системи керування. Як правило, АСУ ТП має єдину систему операторського управління технологічним процесом у вигляді одного або декількох пультів управління, засобу обробки і архівації інформації про хід процесу, типові елементи автоматики: датчики, облаштування управління, виконавчі пристрої. Для інформаційного зв'язку усіх підсистем використовуються промислові мережі.

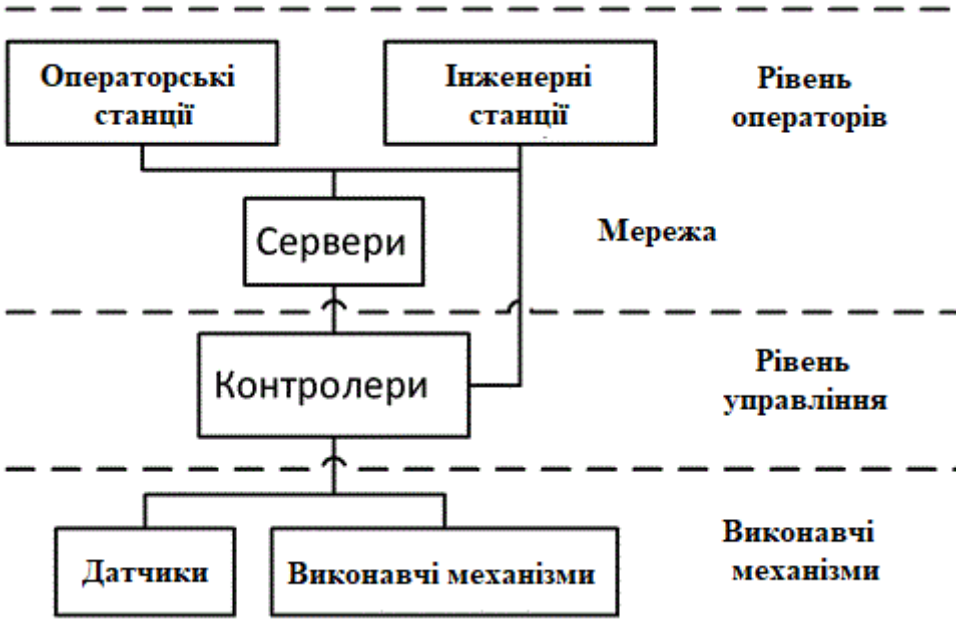


Рисунок 1.4 Структура DCS систем.

Функціональні модулі класу MES - спеціалізоване прикладне програмне забезпечення, призначене для вирішення завдань синхронізації, координації, аналізу і оптимізації випуску продукції у рамках якого-небудь виробництва. Як що порівняти MES та ERP системи, ми побачимо, що MES системи впроваджують керування в основному технологічними процесами. Натомість ERP система містить у собі функціонал звичний для MES систем та має

інструменти для керування та обробки фінансових та/або управлінських завдань.

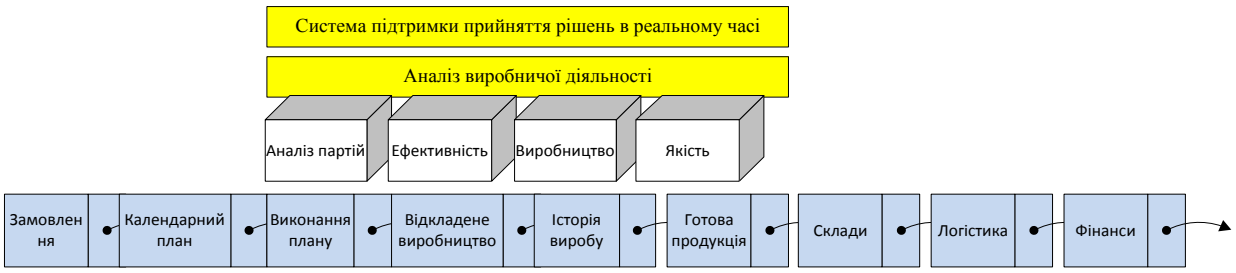


Рисунок 1.5 Структура MES систем.

На думку сучасних експертів MES або SCADA системи не є достатніми для досягнення мети автоматизації виробництва. Лише сучасні комплексні ERP системи дозволяють запровадити спільну обробку інформації та автоматизацію усіх рівнів. Термін ERP, що прийшов до нас із заходу, може бути розтлумачений досить різноманітно, і частенько під ERP приймають системи зовсім іншого класу. ERP - це комплексна система управління бізнес-процесами підприємства [1]. ERP системи розробляються для всебічного керування підприємством в таких аспектах як виробничі процеси, планування ти керування фінансами, кадри ,складське господарство.

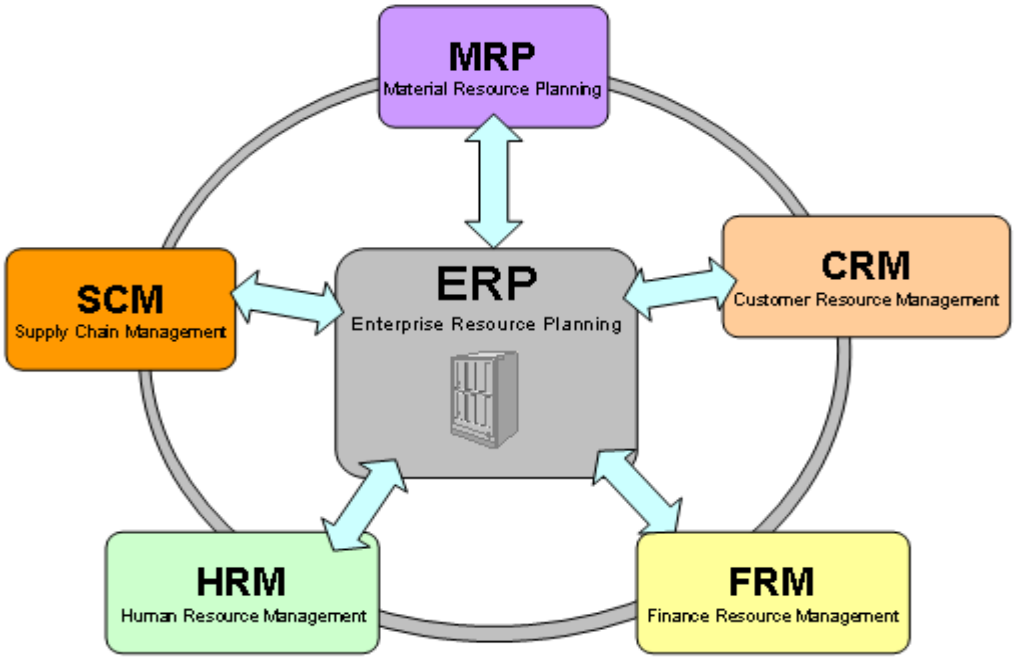


Рисунок 1.6 Структура ERP систем.

Ключове завдання програмних продуктів класу ERP є об'єднання інформаційного простору підприємства, це надає підрозділам можливість зручно та швидко обмінюватися інформацією. Оперативні дані які отримуються в режимі реального часу та є прозорими дозволяють керівництву приймати ефективні та зважені рішення. Будь яка комерційна структура започатковується для досягнення економічного ефекту, ERP системи допомагають аналізувати економічну доцільність бізнес-процесів усередині підприємства та планувати наступні кроки. Це робиться завдяки координації процесів між відділами, структуризації та оптимізації складних взаємозв'язків. Аналіз попередньої діяльності надає можливість частково передбачати наслідки прийнятих рішень та оптимізує витрати. ERP істотно скорочує необхідність та час узгодження дій між структурами всередині підприємства. База даних, що використовується в ERP системі містить у собі інформацію повного циклу роботи підприємства, одночасний доступ усіх виконавців та керівників скорочує час від прийняття рішення до його виконання. Такі системи охоплюють обсяги роботи MES систем та й не тільки. Зокрема керування життєвим циклом продукції або наприклад керування ланцюжками постачання чи політики стосовно дилерських мереж. Також ERP можуть містити управління показниками якості чи формування план-графіків MRP2. Компанія Forrester Research стверджує - майбутнє розробки ERP систем це насамперед розробка керована соціальними чинниками. Великі софтверні компанії вкладають мільярди в розробку інструментарію для ERP.

На сьогодні будь який додаток, що дозволяє обробляти одночасно кількома користувачами великі обсяги інформації, по суті є GUI (Graphical user interface [3]*)* до бази даних. В нашому випадку до Microsoft SQL 2017 Standard. Так наприклад додатки 1С:Підприємство, MS Project та ін. формують команди до СКБД (Системи керування Базами даних) що до збереження, редагування, пошуку або видалення користувацьких даних у таблицях в середині бази даних.

Для подібних завдань використовується додаток 1С:Підприємство.

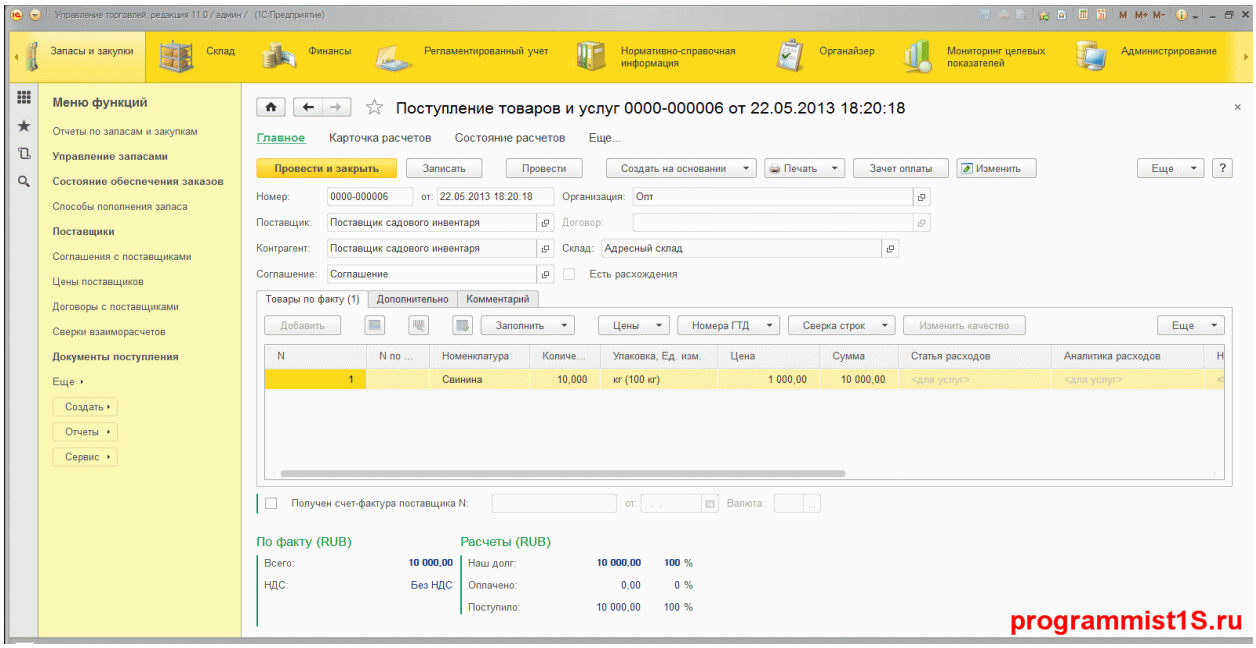


Рисунок 1.7 Приклад інтерфейсу 1С:Підприємство.

№ п/п	Вид продукции	Код вида продук-ции	Сведения о производителе / импортере: наименование производителя/ импортера	ИНН	КПП	Остаток на начало отчетного периода	Поступления: закупки: в том числе: от организа-ций-про-изводи-телей	от организа-ций оптовой торговли	по импорту	итого	возврат продукции	прочие поступ-ления
А	1	2	3	4	5	6	7	8	9	10	11	12
	Сведения по организации											
1	Водка	200	Ивкульский спиртзавод	0123001121	012301001	7 818,05000		7 500,00000		7 500,00000		
2	Водка	200	Аркадакский вино - водочный завод ООО			504,40000						
3	Коньяк и арманьяк, реализуемые в бутылках	229	Торговый дом "Мир вина"	7705260699	770501001	1 002,50000						50,00000
4	Вино (виноградное)	401				2 003,00000						

Рисунок 1.8 Приклад інтерфейсу 1С:Підприємство.

Він був розроблений для задоволення потреб широкого спектру підприємств та організацій, має внутрішню мову програмування для здійснення модифікацій інтерфейсу. Це надає можливість використовувати розгалужену франчайзингову мережу, що здійснює перепродаж, налагодження та супровід продукту.

Додаток 1С:Підприємство надає можливість вести ділову документацію в багатьох аспектах таких як: бухгалтерія, складське господарство, торгівля,

кадрові питання, облік податків, банківські перекази. Але в ньому немає оптимізації для вузькоспеціалізованих виробництв. Тобто при виробництві де ми маємо початкові дані та кінцеві, складно налагодити цикли, де наявні лабораторні досліди та корекція результатів. Також 1С:Підприємство не є системою, яка б дозволила використовувати наявний досвід, або пошук рішень по базі довідникової інформації.

MS Project є додатком який реалізує відсутнє в 1С:Підприємстві, він призначений для керування проектами, контролю за своєчасним виконанням ключових показників.

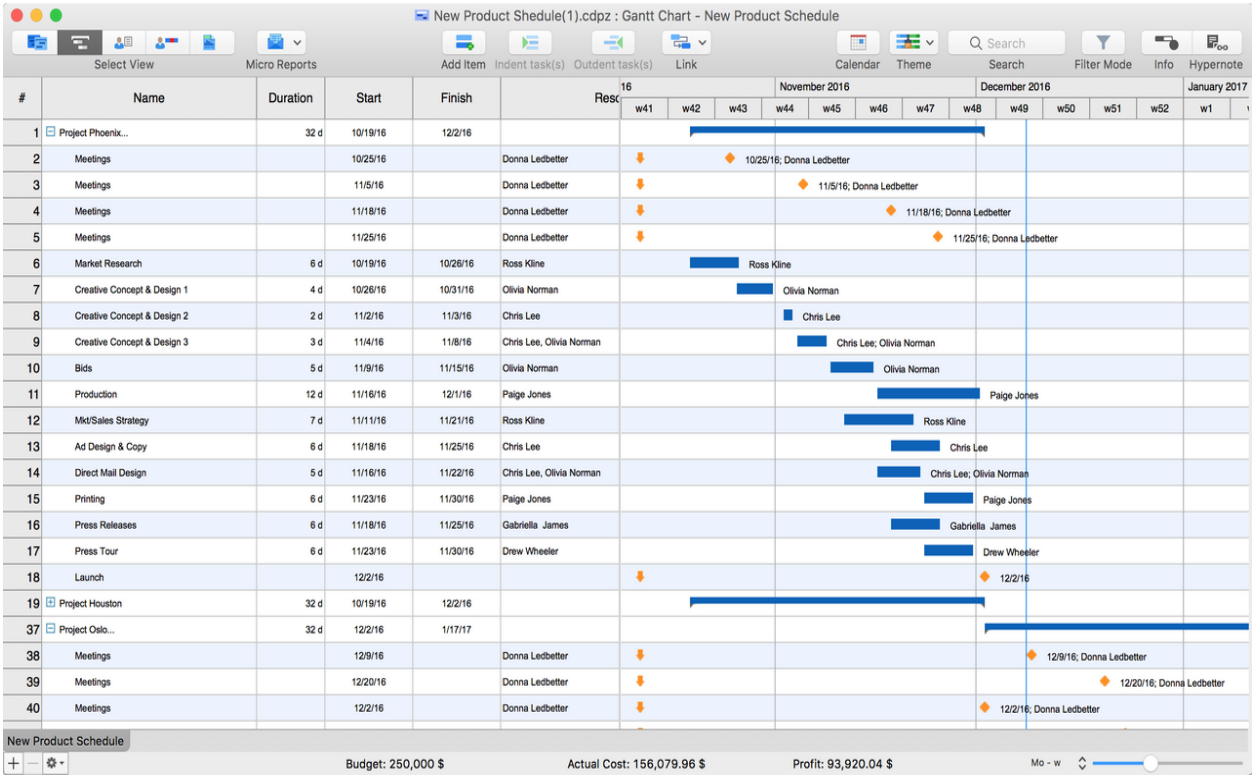
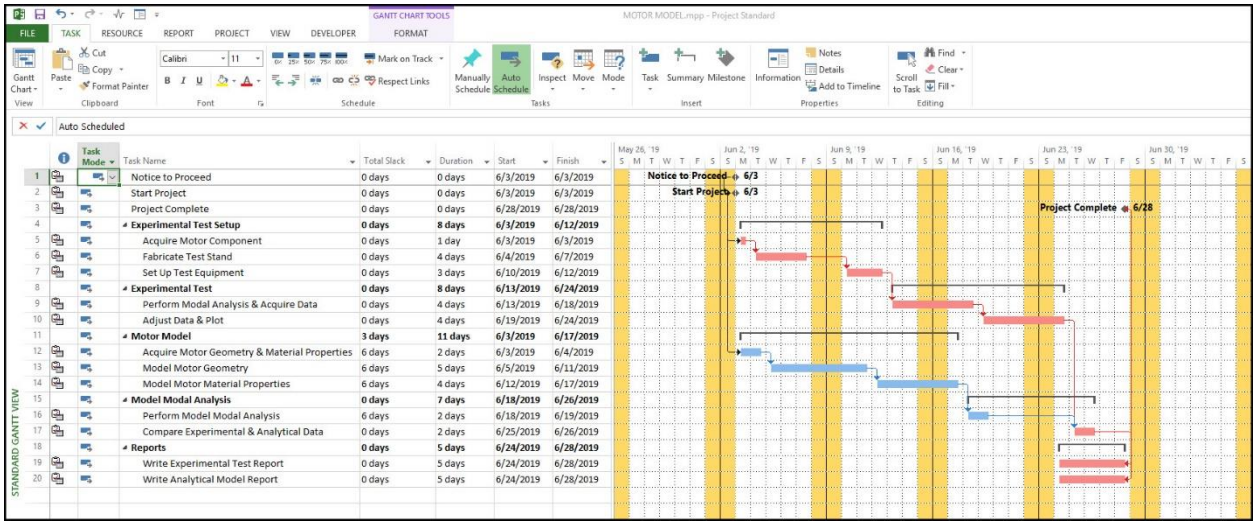


Рисунок 1.9 Приклад інтерфейсу MS Project.



	Task Mode	Task Name	Total Slack	Duration	Start	Finish
1		Notice to Proceed	0 days	0 days	6/3/2019	6/3/2019
2		Start Project	0 days	0 days	6/3/2019	6/3/2019
3		Project Complete	0 days	0 days	6/28/2019	6/28/2019
4		**Experimental Test Setup**	**0 days**	**8 days**	**6/3/2019**	**6/12/2019**
5		Acquire Motor Component	0 days	1 day	6/3/2019	6/3/2019
6		Fabricate Test Stand	0 days	4 days	6/4/2019	6/7/2019
7		Set Up Test Equipment	0 days	3 days	6/10/2019	6/12/2019
8		**Experimental Test**	**0 days**	**8 days**	**6/13/2019**	**6/24/2019**
9		Perform Modal Analysis & Acquire Data	0 days	4 days	6/13/2019	6/18/2019
10		Adjust Data & Plot	0 days	4 days	6/19/2019	6/24/2019
11		**Motor Model**	**3 days**	**11 days**	**6/3/2019**	**6/17/2019**
12		Acquire Motor Geometry & Material Properties	6 days	2 days	6/3/2019	6/4/2019
13		Model Motor Geometry	6 days	5 days	6/5/2019	6/11/2019
14		Model Motor Material Properties	6 days	4 days	6/12/2019	6/17/2019
15		**Model Modal Analysis**	**0 days**	**7 days**	**6/18/2019**	**6/26/2019**
16		Perform Model Modal Analysis	6 days	2 days	6/18/2019	6/19/2019
17		Compare Experimental & Analytical Data	0 days	2 days	6/25/2019	6/26/2019
18		**Reports**	**0 days**	**5 days**	**6/24/2019**	**6/28/2019**
19		Write Experimental Test Report	0 days	5 days	6/24/2019	6/28/2019
20		Write Analytical Model Report	0 days	5 days	6/24/2019	6/28/2019

Рисунок 1.10 Приклад інтерфейсу MS Project.

Цей додаток має необхідні інструменти для контролю критичних до часу операцій, надає можливість планувати час підрозділів та співробітників. Діаграма Ґанта як інструмент візуалізації допомагає керівнику корегувати плани та оцінювати ефективність виконання робіт.

Тож проведемо аналіз можливості використання цих додатків на підприємстві.

Таблиця 1.1 Порівняння 1С:Підприємство та MS Project.

	Календарне планування	Розрахунок матеріалів	Розрахунок собівартості	Планування персоналу
1С:Підприємство	Є частково	Є повністю	Є повністю	Є частково
MS Project	Є повністю	Немає	Немає	Є повністю

Як ми бачимо обидва додатки мають сильні та слабкі сторони, та мали б доповнювати одне одного в умовах підприємства ДП «Завод ОБ та ВТ».

Але було вирішено розробляти окремий додаток, який би був оптимізований до потреб виробництва. Окрім зазначених у табл.1.1 параметрів також необхідно було реалізувати: перевірки на відсутність конфліктів по використанню обладнання, контроль строків виконання, облік енергоресурсів, наявність бази знань що до нетипових ситуацій та методів їх усунення. При

розробці додатку важливо розподілити доступ до інформації та чітко диференціювати інтерфейси користувачів. Було вирішено зробити це на базі посадових обов'язків. Наприклад ливарник додає та редагує лише дані що стосуються його частини технічного процесу, він не в змозі бачити та коригувати інші частини інформації.

Розробка додатку ведеться відповідно до стандартів підприємства - скорочено СТП та координується начальником відділу стандартизації. Із замовником було узгоджено виготовлення вихідного коду (source) за допомогою інтегрованого середовища швидкої розробки програмного забезпечення Delphi. Цікаво те, що середовище Delphi було з початку розроблено для побудови додатків, які працюють з базами даних. На момент створення середовища, найпопулярнішим пакетом баз даних був Oracle database, тож в назві Delphi є гра слів "if you want to talk to [the] Oracle, go to Delphi" [15].

1.2 **Постановка задачі**

За результатами проведеного аналізу існуючих інформаційних систем автоматизації виробництва, для розробки інформаційної системи автоматизації виробництва бурильних труб поставлені такі задачі:

1. Вибрати ПЗ та середовище розробки інформаційної системи.
2. Розробити ПЗ інформаційної системи.
3. Протестувати розроблену інформаційну систему.

2 ВИБІР МЕТОДУ РІШЕННЯ.

2.1 Вибір засобів програмування.

Оскільки проект передбачає створення гнучкого програмного продукту, було необхідно формально розділити дані та функції ПЗ. Для цього зазвичай використовується Об’єктно орієнтоване програмування (ООП). Нині кількість прикладних мов програмування (список мов), що реалізовують об'єктно-орієнтовану парадигму, є найбільшою по відношенню до інших парадигм. Найбільш поширені в промисловості мови (C++, Delphi, C#, Java та ін.) утілюють об'єктну модель Симули. Прикладами мов, що спираються на модель Смолтока, є Objective - C, Python, Ruby.

Слід зазначити, що окрім об’єктно орієнтованого програмування іноді використовується процедурне та функційне. Процедурне програмування - програмування на імперативній мові, при якій послідовно виконувані оператори можна зібрати в підпрограми, тобто більші цілісні одиниці коду, за допомогою механізмів самої мови. Функційне програмування наголошує на застосуванні функцій та уникає змін у послідовності виконання [7]. Об'єктно-орієнтоване програмування (ООП) це програмування методом представлення програми як сукупності об'єктів. Кожен об’єкт є представником певного класу. Класи у свою чергу є частиною ієрархії спадкоємства. Узагальнюючи можна сказати, що ООП це принцип програмування моделей об'єктів інформації, покликаний вирішити основне завдання структурного програмування. При реалізації великих проектів важливим є критерій керованості, яка досягається шляхом структуризації інформації згідно з концепцією ООП.

Керованість в ієрархічних системах фактично означає нормалізацію даних або мінімізацію надмірності та механізм контролю цілісності даних. Тож через керованість отримуємо можливість стисло та чітко транслювати інформацію в зручній для використання формі.

Основні принципи структуризації у разі ООП пов'язані з різними аспектами базового розуміння предметного завдання, яке потрібно для оптимального управління відповідною моделлю:

абстракція (походить від латинського abstrahere — «відволікати») дозволяє уникнути обробки нікчемних у даному випадку атрибутів предмету та натомість виділити важливе, що формалізується як клас;

інкапсуляція для впровадження ієрархічної керованості. Це якість системи, що дозволяє поєднати вхідні дані та методи які з ними працюють в клас (дозволяє задавати команду на обробку без уточнення методів). Спадкоємство надає можливість описувати новий клас на основі вже існуючого з частково або повністю співпадаючими властивостями. Клас від якого походить спадкоємство називають базовим або батьківським, до якого йде спадкоємство є похідним класом або класом-нащадком;

поліморфізм для визначення точки, в якій єдине управління краще розпаралелювати або навпаки, - зібрати воєдино. Тобто фактично йдеться про використання об’єктів з однаковим інтерфейсом без інформації про тип або внутрішню структуру. Коротко, поліморфізм надає спроможність об'єкту використовувати методи похідного класу, якого ще немає на момент створення базового класу. Тобто створюється абстрактний клас в якості шаблону, в якому ми вказуємо визначені на цей час властивості. Ця частина коду відтепер буде повторно використана стільки разів, скільки похідних класів треба буде створити. В цілому об'єктна орієнтація робить програмування ближчим до звичайних людських мов спілкування.

2.2 Основні поняття.

Абстракція даних.

Уявіть собі що ви кермуєте автівкою по вулицях міста. На пішохідному переході ви бачите людину, ваш мозок не аналізує вік, зріст, вагу або стать. Він просто приймає рішення пригальмувати автівку, бо в даному випадку єдина важлива властивість є те, що це пішохід. Тож властивість виділяти лише важливі атрибути об'єкту це абстракція.

Інкапсуляція.

Інкапсуляція - властивість системи, що дозволяє об'єднати дані і методи, працюючі з ними, в класі та приховати від користувача як це реалізовано.

Спадкоємство.

Спадкоємство дозволяє описати новий клас на основі вже існуючого при цьому частково або повністю запозичивши його властивості. Клас, від якого робиться спадкоємство, називається базовим або батьківським класом. Новий клас – нащадком, спадкоємцем або похідним класом.

Поліморфізм підтипів.

Поліморфізм підтипів часто називається просто поліморфізм. Суть поліморфізму в використанні спадкоємства для створення нових класів з використанням тільки інтерфейсів без використання внутрішньої структури. Також під терміном узагальнене програмування часто мається на увазі параметричний поліморфізм.

Клас.

Клас це абстракція даних, яка моделює деяку суть та методи взаємодії з нею. Клас описується набором "полів" та "методів". Клас може описувати будь яку множину предметів, наприклад "фрукти" це абстрактне поняття і це є клас, в нього визначені загальні властивості. Якщо є опис властивостей класу, з'являється можливість створити екземпляр класу, наприклад "яблуко" і це вже буде об'єкт. "Поля" містять загальні властивості, "методи" містять опис як з об'єктами класу можна взаємодіяти. Тобто "методи" фактично є описом властивостей інтерфейсу. Якщо ми копіюємо властивості класу методом

привласнення, в такому випадку копіюються тільки "методи", тобто інтерфейс, але не внутрішні дані. Змінна-об’єкт, що відноситься до заданого класом типу, є екземпляром цього класу. При розробці додатків класи створюються так, щоб забезпечити відповідність природі об'єктів, цілісності даних та простий інтерфейс. Спадкоємство забезпечує цілісність предметної області об'єктів, їх інтерфейсів та зручність проектування. Зазвичай для зручності в розробці використовуються блоки з методами, які відповідають за привласнення та зчитування значення. Ці блоки називаються "властивостями" та частіше за все робляться майже співпадаючими за ім'ям з "полями". В деяких програмах клас може оброблятися як об'єкт.

Об’єкт.

Об'єкти можна поділити на категорії. Реальні об'єкти як абстракції реально існуючих предметів, істот тощо. Ролі - абстракція мети, посади, виконуваної роботи. Інциденти це абстракція будь якого випадку або дії, наприклад землетрус або повінь. Взаємодії - об'єкти похідні від взаємодії інших об'єктів, наприклад договір між контрагентами. Специфікації які можуть бути використані для опису критеріїв або правил. Тобто об'єкт це абстракція, що з'являється при створенні екземпляру класу.

Ключові поняття:

1. квантифікація змінних типу (універсальна, екзистенціальна, обмежена);
2. підтипізація (англ. subtyping - стосунки "супертип-підтип"); включення (англ. subsumption - окремий випадок підтипізації).
3. об'єктний тип на відміну від типу "запис" містить не тільки поля даних, але й методи опис яких є в описі об'єктів. Об'єктні типи можуть бути побудовані за допомогою спадкоємства на базі класів додаванням приватних полів, функцій та процедур. Значення об'єктного типу яке є екземпляром класу породжується конструктором за початковими параметрами.

2.3 Класифікація.

За основними концепціями ООП можна стпробувати класифікувати більшість об'єктно орієнтованих мов. Природньо це теоретичне обґрунтування не є істиною в останньої інстанції. Різні джерела тлумачать концепції ООП з великим розкидом варіантів. Більш відповідними до концепції є мови нащадки Алголу та Симули. Менш відповідними є мови ідеологічно похідні від Smalltalk,так наприклад принцип приховання в Smalltalk не реалізовано тому що вважається несуттєвим. Наприклад в мовах типу Haskell, OCaml, SUML, Mondrian концепції ООП не тільки підтримуються в більш значній мірі, але й не вимагають підтримки з боку мови та можуть бути емульовані. На відміну від імперативного програмування, основні ОО-мови мають за мету підвищення коефіцієнту повторного використання коду. Мови що містять механізм поліморфної типізації з застосуванням концепцій ООП, мають трохи нижчий показник повторного використання коду. Це викликано переходом від параметричного до додаткового (ad-hoc) поліморфізму. У деяких мовах коефіцієнт повторного використання коду дуже залежить від дисципліни програміста. На приклад це мови з динамічною типізацією такі як Python або Ruby та ін. в них принципи ООП використовуються для побудови логічної структури програми. В традиційних ОО-мовах таких як C, C#, C++ , SQL, T-SQL частіше використовується не структурна а номінативна типізація. Хоча вона є менш гнучкою за структурну, але більш безпечною. Типи даних є сумісними лише тоді, коли властивості типів посилаються на той самий тип, або спільна обробка класів можлива лише за умови, що ці класи є похідними (спадкоємцями) від одного класу. Мови що використовують поліморфну типізацію, частише характерізуються узгодженням типів змінних за їх структурою. Тобто використовується структурна типізація, якщо у змінних співпадає набір атрибутів, ці змінні важаються сумісними.

Також є мови наприклад Python, Objective-C, Ruby що використовують динамічну типізацію. Вони займають проміжну позицію. При динамічній типізації значення змінних перевіряються не під час компіляції а під час

виконання програми, тому можуть мати будь які значення. Джузеппе Кастанья в 90х роках минулого сторіччя узагальнено обґрунтував реалізацію поліморфізму методом динамічної диспетчеризації. Плюсом цього методу, що при створенні програми можна описувати дії над змінними яких ще немає. В мінуси можна зарахувати збільшення обчислювальних ресурсів необхідних для виконання програми та більша ймовірність помилок які залишаться після компіляції. Мови в яких реалізовано динамічну типізацію найбільш орієнтовані на роботу з даними змінних типів. Напрямком подальшого розвитку ООП є компонентно орієнтоване програмування та подієво орієнтоване програмування. Є передумови для появи в майбутньому агентно орієнтованого програмування, в якому агенти є частинами коду на рівні виконання, що спілкуються за допомогою зміни середовища. В традиційному ООП взаємодія об'єктів відбувається завдяки повідомленням.

У випадку аспектно-орієнтованого програмування використовуються супутні до об'єктів конструкції які не є частиною об'єктів, але потрібні для ефективної та безпечної роботи. Вони інкапсулюються в аспекти та використовуються для виняткових ситуацій або перевірок. Суб'єктно-орієнтоване програмування це подальший розвиток ООП, в ньому класу необхідно вказувати не методи необхідні для досягнення результатів, а результати яких необхідно досягти. При цьому формується не залежна суб'єктна ієрархія. Суб'єкт сам обирає метод для досягнення результатів. Узагальнено СОП це метод в якому ОО системи структуруються як композиція суб'єктів. Першою мовою програмування, в якому були запропоновані основні поняття, що згодом склалися в парадигму, була Симула, але термін "об'єктна орієнтованість" не використовувався в контексті використання цієї мови. У момент його появи в 1967 році в нім були запропоновані революційні ідеї: об'єкти, класи, віртуальні методи та ін., проте це усе не було сприйнято сучасниками як щось грандіозне. Фактично, Симула була "Алголом з класами", що спрощує вираження в процедурному програмуванні багатьох складних концепцій. Поняття класу в Симулі може

бути повністю визначене через композицію конструкцій Алголу (тобто клас в Симулі - це щось складне, описуване за допомогою примітивів). Погляд на програмування "під новим кутом" (відмінним від процедурного) запропонували Алан Кей і Ден Інгаллс в мові Smalltalk. Тут поняття класу стало основною ідеєю для усіх інших конструкцій мови.

2.4 Концепції.

Есенцією ООП є поняття об'єкту. Об'єкт - це суть, яка може посилати повідомлення та має можливість на них реагувати, використовуючи свої дані. Об'єкт є екземпляром класу. Дані об'єкту приховані від іншої програми. Інкапсуляція включає приховування.

Інкапсуляція є достатньо важливою для визначення об’єктності мови програмування, але лише наявність спадкоємства визначає об'єктну-орієнтованість. З позиції ООП, наявність інкапсуляції і спадкоємства не робить мову програмування сповна об'єктною. Істотною перевагою ООП при умові реалізації поліморфізму підтипів є сумісність об'єктів, тобто можливо обробляти однаково різні об'єкти як що у них є загальні інтерфейси.

ООП налічує вже більш ніж сорокарічну історію, незважаючи на це, досі не існує чіткого визначення цієї технології. Основні принципи, закладені в перші об'єктні мови і системи, були піддані істотній зміні (чи спотворенню) і доповненню при численних реалізаціях подальшого часу. Окрім цього, десь із середини 1980-х років термін "об'єктно-орієнтований" став трендовим, згодом з ним сталося те ж саме, що дещо раніше з терміном "структурний" - його стали штучно "додавати" до геть усіх нових розробок, щоб забезпечити їм додаткову привабливість. Бьєрн Страуструп, автор мови програмування C++ та дійсний член "Texas Academy of Medicine, Engineering, and Science", зауважував на відсутності чітко окреслених ознак об'єктної орієнтованості. Багато джерел використовували формулу: "X - це добре. Об'єктна орієнтованість - це добре. Отже, X є об'єктно-орієнтованим". За словами Тімоті Бадда, Роджер Кінг аргументовано наполягав, що його улюблений кіт

об'єктно-орієнтований. Окрім інших своїх переваг, кіт репрезентує характерну поведінку, виявляє реакцію на повідомлення, має успадковані реакції та керує своїм, окремим, внутрішнім станом. Алан Кей, який є творцем мови Smalltalk, та вважається одним з піонерів ООП, переконував що, об'єктно-орієнтований підхід міститься в такому наборі основних принципів. Все є об'єктом - обчислення виконуються за допомогою обміну даними (взаємодії) між об'єктами, який є вимогою одного об’єкту, щоб інший об'єкт виконав деяку дію. Взаємодія об’єктів зводиться до посилки або отримання повідомлень. Повідомлення являє собою запит на виконання дії доповнений комплектом аргументів, що можуть знадобитися при виконанні дії [4]. Кожен об'єкт має незалежну пам'ять складену з інших об'єктів.

Будь-який об'єкт має виділену пам'ять складену з інших об'єктів. Кожен об'єкт є представником класу, який має загальні властивості об'єктів (наприклад int або varchar). У класі задається поведінка (функціональність) об'єкту. Отож усі об'єкти, що є екземплярами одного класу, мають можливість виконувати такі самі дії. ієрархією спадкоємства є класи організовані в єдину деревовидну структуру із загальним коренем. В ООП програма складається з об'єктів, які мають стан та поведінку. Взаємодія між об'єктами забезпечується повідомленнями, що в свою чергу теж є об'єктами. Сама програма є об'єктом. Тобто об'єктно-орієнтована концепція узагальнено виглядає так - будь-що є об'єктом. Всередині програми об'єкти формують ієрархію, пам'ять та поведінка екземплярів батьківського класу автоматично доступна екземплярам класів спадкоємців, це стосується й опосередкованого спадкоємства. Проблему виникнення нескінчених рекурсій було вирішено на рівні середовища/мови програмування шляхом перетворення повідомлень до об'єктів в повідомлення до стандартних системних об'єктів. Розподілення функціональних обов'язків об'єктів істотно підвищує керованість та стійкість системи вцілому. Також не останню роль в ціому відіграє чітке визначення інтрефейсів для взаємодії об'єктів та інкапсуляція. Але існують й інші ознаки, за якими можна визначити об'єктну орієнтованість. Окреме поняття класу в

ООП обумовлене жагою до можливості класифікувати різні об’єкти за ознаками поведінки. Клас в ООП це абстрактний тип даних, що створюється програмістом. За цією тезою усі об'єкти є значеннями цього абстрактного типу, а внутрішня структура значень, набір операцій, які можуть бути виконані над ними визначається за класом. Спадкоємство надає можливість повторно використовувати код – в тому разі, якщо декілька класів мають схожу поведінку, немає сенсу дублювати їх опис, оптимальним буде сформувати зі споріднених частин загальний батьківський клас, а описом окремих класів зробити саме ті елементи, що відрізняються. Для підтримки можливості записувати різні об'єкти в однотипні змінні потрібен поліморфізм. Об'єкт генерує повідомлення, при цьому не вказується клас адресата. Повідомлення, що є ініційованими однотипними змінними можуть містити у собі об'єкти різних класів та можуть викликати різну реакцію. Обмін повідомленнями як поняття може бути інтерпретований так: первинна взаємодія об'єктів представлялася як "реальний" обмін повідомленнями. Як приклад в мові Smalltalk виконувалось відправлення спеціального об'єкту "повідомлення" між різними об'єктами. Для опису паралельних обчислювань можна використати модель де кожен активний об'єкт має виділений потік виконання, поводиться як незалежний обчислювальний процес, працює в той самий час що й інші активні об'єкти. Така модель є узагальненою.

2.5 Особливості реалізації

В таких мовах програмування як Objective - C, Python, Ruby, Smalltalk, в поліморфних змінних код класу до якого відноситься об'єкт, що належить до змінної, обробляється повідомленням без урахування того, як була оголошена зміна. Недоліком цього методу є те, що використання спільного механізму обміну повідомленнями збільшує витрати обчислювальних ресурсів. Так обробка поліморфних змінних за допомогою обміну повідомленнями є гнучким інструментом, але потребує додаткових ресурсів. У багатьох

сучасних ОО-мовах, наприклад в Java використовується концепція "виклик методу через повідомлення". Створимо метод який можна викликати зовні:

```
public static int methodName(int n, int m) {
  // метод
}
```

в цьому прикладі public static — модификатор (static означає, що для звернення до методу немає необхідності вказувати клас); int — тип даних, які будуть повернені; methodName — ім'я методу; n, m — формальні параметри; int n, int m — перелік параметрів.

Використання такої концепції робить повідомлення несамостійними, вони не мають атрибутів. Це значно обмежує можливості програмування. Деякі мови використовують гібридне представлення, демонструючи переваги одночасно обох підходів - наприклад, CLOS, Python. Концепція віртуальних методів покликана вирішити питання виконання потрібних методів з поліморфними змінними. Віртуальний метод, це метод класу, що може бути змінений в похідних класах, тож що міститься в цьому методі буде з'ясовано вже при виконанні. Подивимось як це реалізовано в C++:

```
class Base
{
public:
   virtual void f() = 0;
};
class Derived : public Base
{
public:
   void f();
};
```

Ми створили оголошення віртуального методу, але не реалізували f() = 0. Тож кожен об'єкт це значення, яке відповідає певному класу. Тобто сучасні ОО-мови можна вирізнити за ознакою чіткої класифікації об'єктів.

Клас є оголошеним програмістом складеним типом даних, який має в собі поля даних та методи. Ці складові можна описати наступним чином:

1. Поля даних

Параметри об'єкту (звичайно не всі, а тільки необхідні в програмі), які задають його стан (властивості об'єкту предметної області). Іноді поля даних об'єкту називають властивостями об'єкту, через що можлива плутанина. Фактично поля є значеннями (змінні, константи), оголошеними як ті, що належать класу.

2. Методи

Процедури і функції, пов'язані з класом. Методи це перелік дій, які можна виконувати над об’єктом, або дій, які сам об’єкт може виконувати. Класи можуть наслідувати один від одного. Клас-нащадок отримує усі поля і методи класу-батька, але може доповнювати їх власними та потрібні змінює. Мови програмування можна поділити за принципом спадкоємства, деякі підтримують лише одиничне, деякі множинне. Якщо клас-нащадок може мати лише один клас-батько, це одиничне спадкоємство, в іншому випадку (коли клас-нащадок має більше одного класу-батька) це вже множинне спадкоємство. Множинне спадкоємство створює не тільки логічні проблеми. Наявні проблеми реалізації заважають поширенню його підтримки. Альтернативою множинному спадкоємству в об’єктно орієнтованих мовах стали інтерфейси. Активне використання інтерфейсів почалося в 90-і роки минулого століття.

Інкапсуляцію можна розглядати як сукупність таких компонентів:

1. Контроль доступу

Для розподілення методів класу на ті, що забезпечують внутрішню логіку роботи об'єктів та ті, які потрібні для взаємодії на зовні, використовується механізм приховання або доступності для внутрішніх або зовнішніх методів. Зони видимості для окремих членів класу окреслюються за допомогою синтаксичних конструкцій наявних в мові програмування.

В більшості ОО-мов використовуються модифікатори public, protected та private. Розглянемо приклад використання модифікаторів в мові C++:

```
class some {
    friend void f(some&);
public:
    int a_;
protected:
    int b_;
private:
    int c_;
};
```

У такий спосіб модифікатор public відкриває доступ для усіх класів, protected доступ тільки для свого класу або класу спадкоємця, private тільки для свого класу.

2. Методи доступу

Для забезпечення керування доступом, поля класу взагалі не мають бути відкриті до модифікації зовні. Неконтрольований доступ дозволив би довільно змінювати внутрішній стан об'єктів. Таким чином є дві реалізації - або мова взагалі немає механізму звернення до полів класу зовні, або поля за замовчанням оголошуються прихованими. Доступ до таких даних забезпечується завдяки спеціальним методам доступу. Методи доступу залежно від дії, яку вони дозволяють зробити називають аксесорами (від англ. access - доступ), геттерами (англ. get - отримати) і сетерами (англ. set - встановити). Якщо потрібно викликати метод доступу, потрібно звернутися до властивостей об'єкту. Вони мають вигляд як звичайні поля доступні для читання та запису, але при зверненні до них викликаються методи доступу.

Таким чином, властивості можна розглядати як "розумні" поля даних, супроводжуючі доступ до внутрішніх даних об'єкту будь-якими додатковими діями (наприклад, коли зміна координати об'єкту супроводжується його перемальовуванням на новому місці). Насправді властивості не додають будь-

яких можливостей, вони використовуються тільки для приховування викликів методів доступу. Можна по різному реалізувати властивості, в різних мовах дуже рідко збігаються засоби реалізації. Тож якщо Delphi оголошення властивості містить у собі лише імена методів доступу необхідні для звернення до поля, натомість у C# безпосередньо вказується код методів доступу. Цей код викликається лише при роботі з властивостями. Такий підхід дозволяє не використовувати додаткових методів доступу для безпосереднього виклику. Самі методи доступу є звичайними методами з деякими додатковими вимогами до сигнатури. Поліморфізм реалізовано введенням в мову правил, згідно з якими змінною типу "клас" може бути присвоєний об'єкт будь-якого класу-нащадка її класу. Пріоритетним застосуванням ООП є розробка складних та великих проектів. Коли є потреба задіяти великий штат програмістів та чітко узгоджувати окремі частини. При розробці великих проектів, коли задіяно декілька або більше програмістів, необхідно чітко документувати структуру додатку. Важливими властивостями є опис наявних частин, їх функції, поведінка та відповідальність. В такому випадку кожен фахівець може, практично не звертаючись до інших за численними консультаціями, розробляти свою частину проекту. Це в свою чергу значно спрощує розподілену розробку, коли різні фахівці можуть робити в різних локаціях або за різним графіком.

Виділення частин робиться так, щоб кожна мала мінімальний за об'ємом і точно певний набір виконуваних функцій (обов'язків), і при цьому максимально можлива кількість взаємодій була всередині, тобто мінімізується зовнішня взаємодія. Подальше уточнення призводить до виділення дрібніших фрагментів опису. У міру деталізації опису і визначення відповідальності виявляються дані, які необхідно зберігати, виявляються передумови до формування класів з агентів за ознакою поведінки. Формуються компоненти та інтерфейси. За умови дотримання технологічної дисципліни з’являється можливість незалежної розробки компонентів.

2.6 Критика.

Велике значення має правильну побудову ієрархії класів. Критики ООП часто згадують проблему крихкості базового класу. Ця проблема стає більш помітна в великих проектах. Ця проблема з'являється тоді, коли розробка майже закінчена або знаходиться на стадії тестування. При спробі зробити будь які зміни в базові класи, змінюються також й похідні класи, це робить систему загалом непридатною. Навіть якщо зміни, що вносяться, не торкнуться інтерфейсу базового класу, зміна його поведінки може непередбачуваним чином відбитися на класах-нащадках. Якщо розробників системи більше одного, на різних етапах розробки система доповнюється класами-нащадками, буває дуже складно вирахувати як вплине на поведінку системи будь-яка зміна базового класу. Для систем в яких ступенів спадкоємства декілька, прогнозувати до чого приведуть зміни базового класу взагалі неможливо [6].

Компонентне програмування як метод вирішення проблеми крихкості базових класів було запропоновано Клаусом Віртом. Компоненти, що компілюються окремо, можуть динамічно підключатись вже під час виконання.

Компонентне програмування.

Компонентно-орієнтоване програмування – базується на ООП, та є надбудовою, набір правил та обмежень, спрямованих на побудову великих програмних систем, що розвиваються, та мають великий життєвий цикл. Програмна система в цій методології є набором компонентів з певними інтерфейсами. Компонентно-орієнтоване програмування дозволяє вносити зміни в існуючі системи за допомогою створення нових компонентів, або змінюючи наявні. Як приклад нижче буде розглянуто оновлення SQL процедури використане при створенні ІС UBTPlan. Якщо виникає потреба розробити новий компонент, використовуючи вже наявний, в цьому випадку застосовується правило, яке забороняє використовувати спадкоємство

внутрішньої структури чи реалізації. Новий компонент може мати лише успадковані інтерфейси. Так розв'язується проблема крихкості базового класу.

Прототипне програмування.

Прототипне програмування, зберігши частину рис ООП, відмовилося від базових понять - класу і спадкоємства. Прототипи використовуються як зразок для створення інших об'єктів. Фактично це є часткове спадкоємство. Тобто об'єкти створені за прототипом можуть бути як незалежними так і пов'язаними з батьківським об'єктом, і в цьому випадку будуть успадковувати зміни в прототипі. Тонкощі реалізації змінюються від мови до мови. Опис класів та походження екземплярів в прототипно-орієнтованих мовах відсутній. Натомість введено поняття створення об'єкту за набором полів даних та методів, які об'єкт повинен містити у собі. Будь-який об'єкт може стати прототипом для створення нового об'єкту. При копіювання об'єкту можуть копіюватися й всі складові даних та методів, або використовуватися посилання на об'єкт-прототип. Такі посилання дозволять об'єкту-клону змінюватись одночасно з об'єктом-прототипом. Вони зберігаються до того часу поки в об'єкті клоні не будуть примусово змінені відповідні поля даних або методи. У випадку використання посилань всередині об'єктів, середовище має містити засіб делегування. Тобто, при зверненні до полів або методів яких об'єкт не містить, повинно відбуватись звернення за посиланням до тих пір поки не буде досягнуто об'єкт який дійсно містить потрібні дані або методи.

Клас-орієнтоване програмування.

Клас-орієнтоване програмування - це програмування, сфокусоване на даних, причому дані і поведінка нерозривно пов'язані між собою. Разом дані і поведінкою є клас. Відповідно в мовах, грунтованих на понятті "клас", усі об'єкти розділені на два основні типи - класи і екземпляри. Клас визначає структуру і функціональність (поведінка), однакову для усіх екземплярів цього класу. Екземпляр є носієм даних - тобто має стан, що міняється відповідно до поведінки, заданої класом. У клас-орієнтованих мовах новий екземпляр створюється через виклик конструктора класу (можливо, з набором

параметрів). Екземпляр, що вийшов, має структуру і поведінку, жорстко задані його класом.

2.7 Інтеграція з СКБД.

ORM (англ. Object - Relational Mapping) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи "віртуальну об'єктну базу даних". Існують як проприєтарні, так і вільні реалізації цієї технології. Необхідно забезпечити роботу з даними в термінах класів, а не таблиць даних і навпроти, перетворити терміни і дані класів в дані, придатні для зберігання в СКБД. Необхідно також забезпечити інтерфейс для CRUD- операцій над даними. Загалом, необхідно позбавитися від необхідності писати SQL- код для взаємодії в СКБД.

Реляційні СКБД

Якщо побудувати взаємні зв'язки між різними типами даних, це вирішує проблему зберігання об'єктно-орієнтованих даних. Тож рішенням є реляційні системи керування базами даних. При використанні реляційних баз потрібно писати додаток у такий спосіб, щоб дані оброблялися в об'єктному вигляді, але потому, зберігалися в реляційній формі. Сервіс Amazon Relational Database Service одна з реалізацій реляційних БД. Реляційна БД репрезентує певну підбірку даних пов'язаних між собою, що зберігаються у вигляді певної кількості таблиць, які в свою чергу складаються з рядків та стовпців. Кожен стовпчик містить певний тип даних, кожна клітина містить значення атрибуту. Кожен рядок таблиці є набором пов'язаних значень відносно до одного об'єкту або суті. Кожен рядок таблиці може бути виділений за допомогою унікального ідентифікатора UID. Цей ідентифікатор називається Primary Key. UID з різних таблиць можуть бути зв'язані за допомогою зовнішніх ключів. Таким чином доступ до даних може бути організований багатьма методами та не потребує перетворення форми даних.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ.

3.1 Інформаційна структура виробництва бурильних труб.

При проектуванні системи було важливо максимально використати наявні структурні, апаратні та програмні засоби. Це дозволить мінімізувати потрібні ресурси та істотно зменшує вартість проекту та час на реалізацію. Була використана наявна магістральна мережа (рис. 3.1)

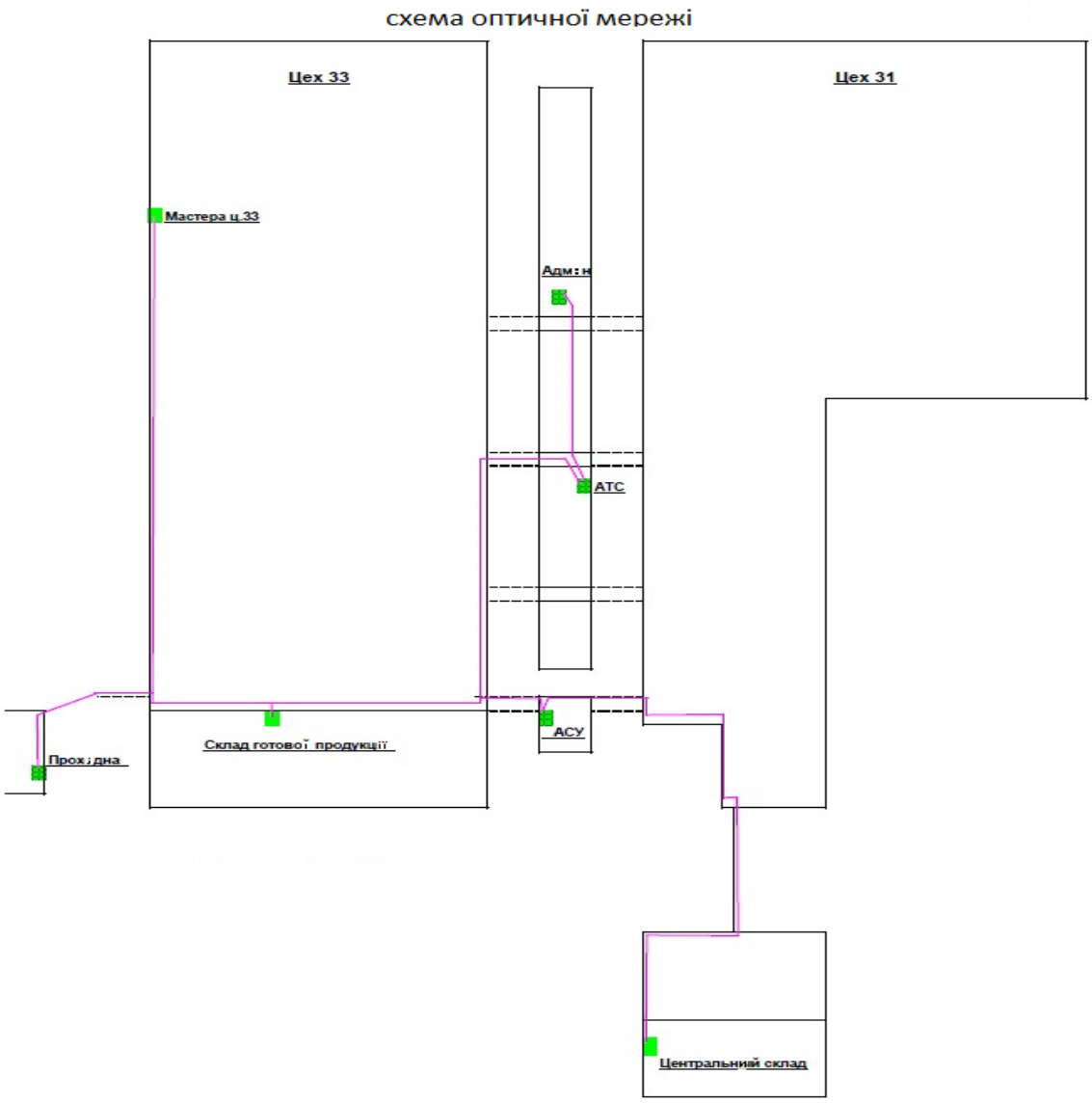


Рисунок 3.1 Схема магістральної мережі.

Та розроблена логічна схема побудови мережі приведена на рис 3.2.

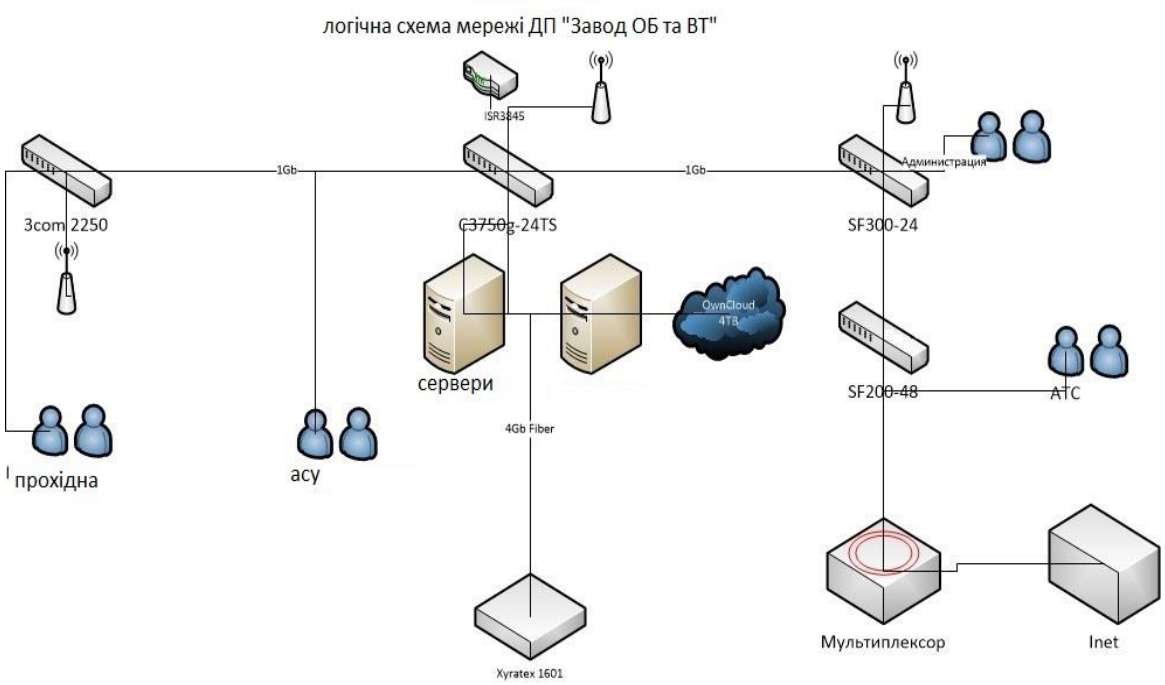


Рисунок 3.2 Логічна будова мережі.

Оскільки UBTplan є комплексним рішенням, він містить у собі й апаратну складову. Для автоматизації підприємства було заплановано побудувати закриту Wi-Fi mesh систему в виробничих приміщеннях. Обладнати робочі місця тонкими клієнтами та терміналами збору даних. Додати ланку захисту інформації, побудовану на базі шлюзу безпеки Cisco ASA 5520 та ПЗ Cisco AnyConnect VPN (рис. 3.3).

Рисунок 3.3 Вигляд Cisco ASA 5520.

З метою підвищення надійності та швидкості відновлення - використати середовище віртуалізації VMWare (рис 3.4,3.5).

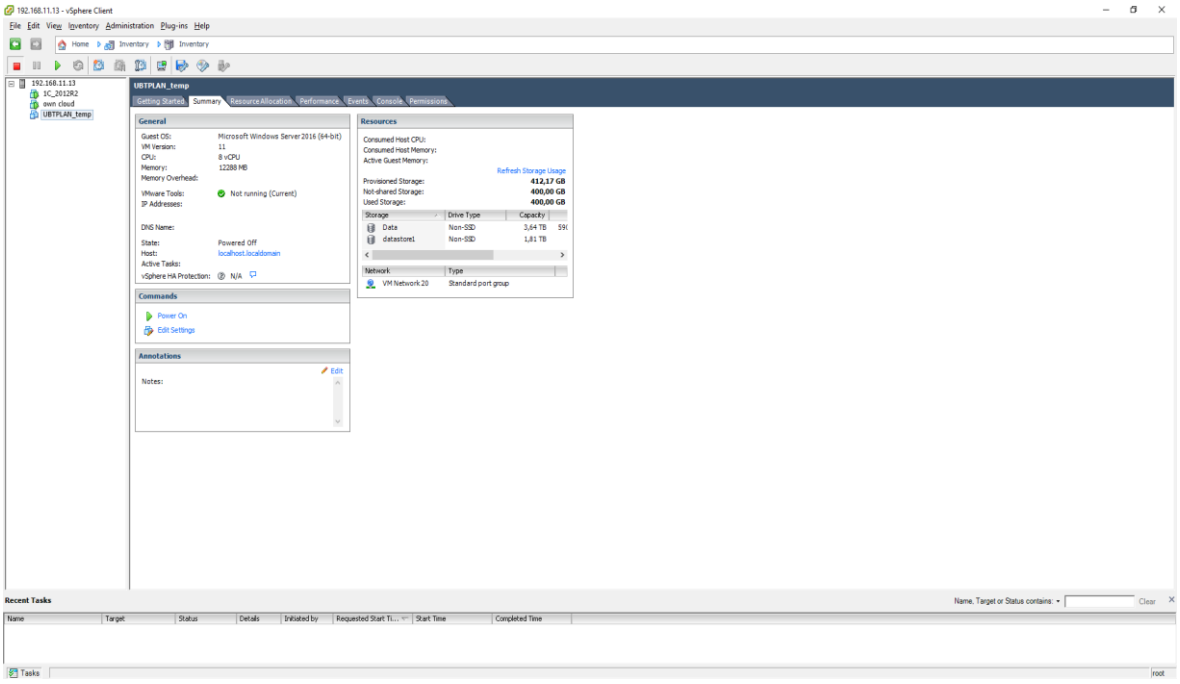


Рисунок 3.4 Створення віртуальної машини в середовищі VMWare.

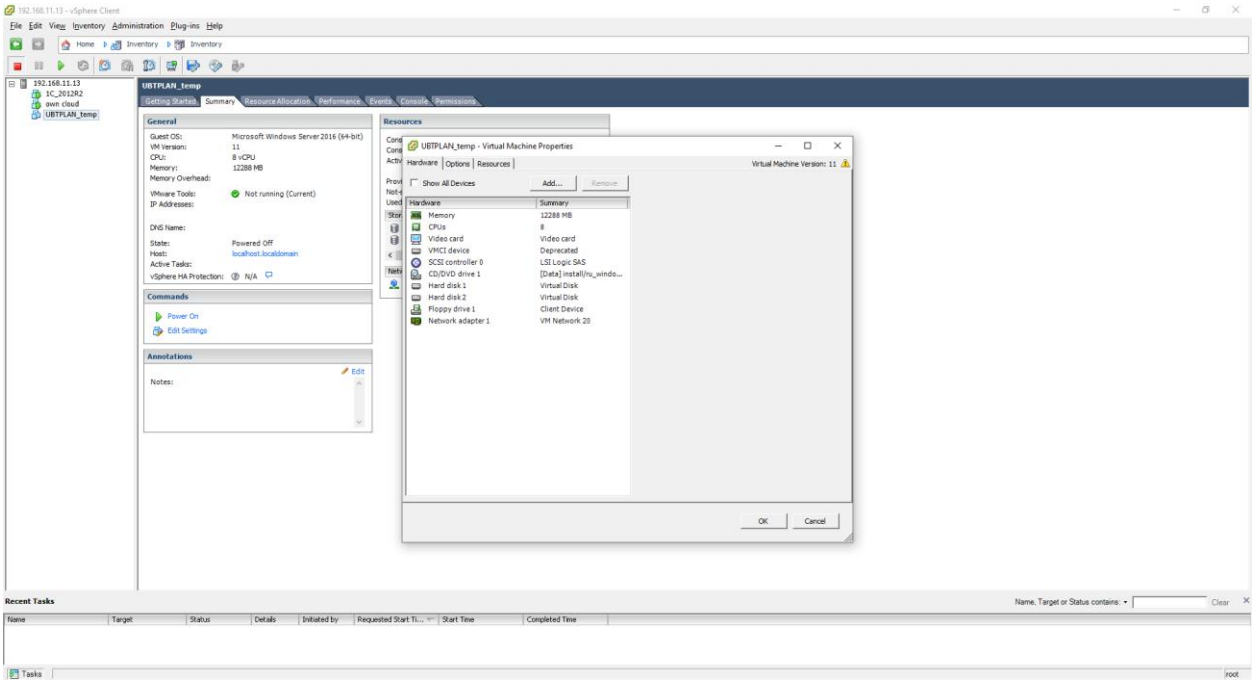
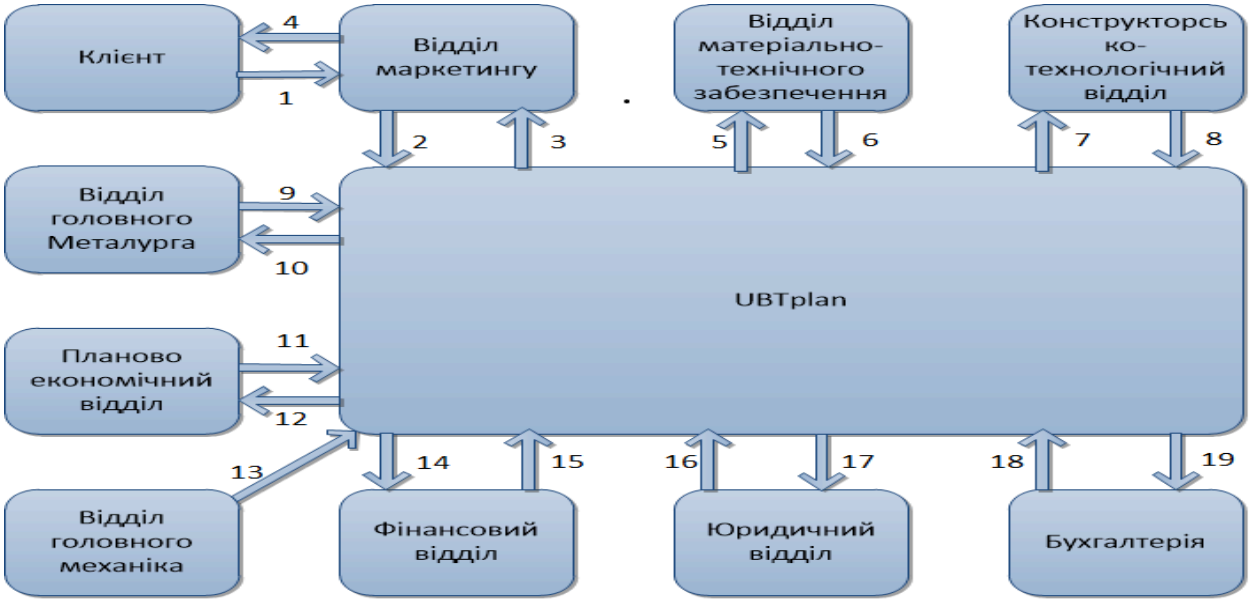


Рисунок 3.5 Налагодження віртуальної машини в середовищі VMWare

3.2 Програмна реалізація інформаційної системи.

Спираючись на технічне завдання було розроблено схему взаємодії підрозділів підприємства. При цьому кожен підрозділ отримав окремі зони відповідальності, що до внесення або корегування інформації. Керівники підрозділів також отримали можливість моніторингу етапів виробництва на замовлення. На рисунку 3.6 зазначені підрозділи, які приймають участь у виконанні замовлення клієнта.



1. замовлення від покупця
2. внесення замовлення до бази
3. отримання цін та умов постачання
4. відповідь покупцю
5. отримання переліку необхідної сировини
6. замовлення на розробку технологічної карти
8,9,13. внесення технічної документації до довідника
10. запит на надання додаткових вимог
11. розрахунок вартості
12,14,15. аналіз та планування
16,17. узгодження договірних умов
18,19. розрахункові операції

Рисунок 3.6 Взаємодія між відділами.

Існує три потоки надходження інформації для внесення в базу даних. Перший, що важливо для складського господарства, це надходження матеріалів для виробництва. Другий - планування технічних процесів та операцій при виробництві конкретних виробів. І третій замовлення від

покупця, в якому він чітко окреслює тип та кількість необхідних виробів. І вже це надає поштовх до закупівлі матеріалів та планування технічних процесів виробництва.

Клієнт надає замовлення на виготовлення виробу. Якщо вироби з такими параметрами вже хоч раз виготовлялися на підприємстві, відділ маркетингу може надати позитивну відповідь на можливість виготовлення замовленого виробу. Також буде врахована собівартість продукції та надано замовлення на закупівлю сировини та планування робочих змін, наприклад ливарників або токарів.

Після того, як клієнт підтвердить замовлення, в системі з’явиться інформація що до черги виробництва виробу. Відповідальні відділи почнуть роботу над забезпеченням сировиною, транспортом, робітниками. Також буде спланована черга для праці на обладнанні, щоб уникнути конфліктів та простоїв.

Оскільки ця система є складною, та має в складі усі аспекти конкретно цього виробництва, інтеграція з іншими програмними засобами не потрібна.

.Схема руху інформації наведена на рис 3.7

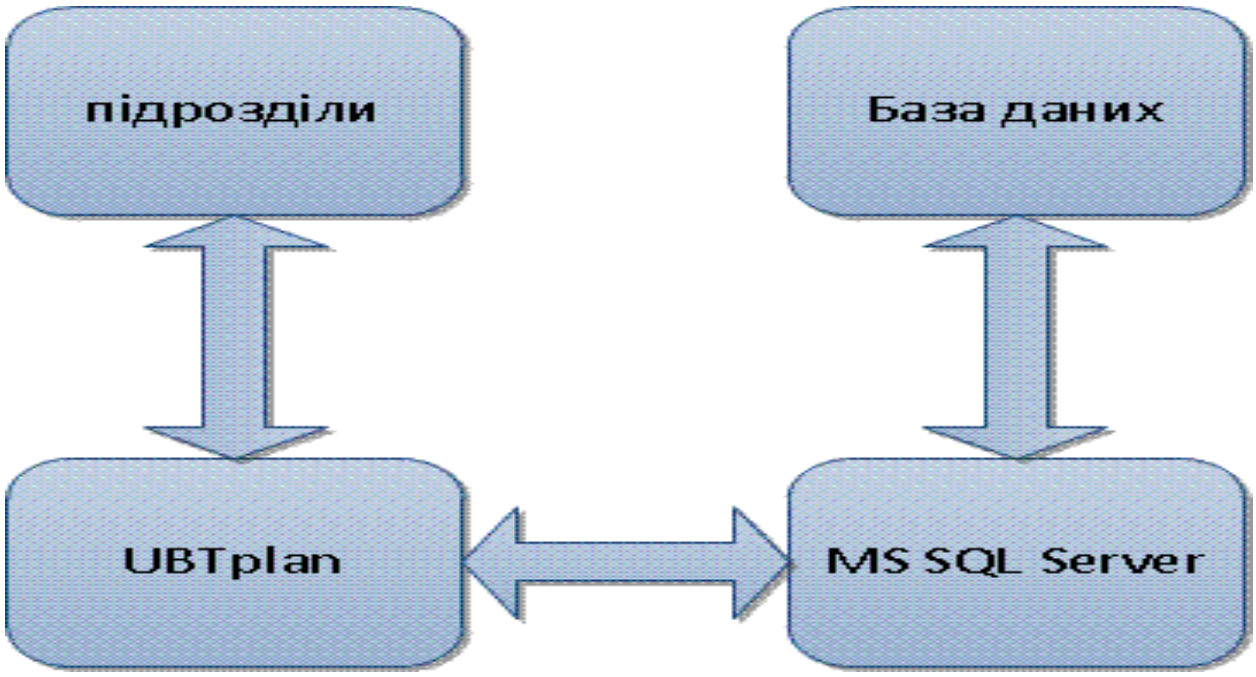


Рисунок 3.7 Схема руху інформації

Додаток працює за класичною схемою він фактично є графічним інтерфейсом між користувачами та СКБД MS SQL.

Базу можна логічно поділити на дві частини: оперативну та довідникову. Оперативна система містить у собі дані про поточне виробництво. Вона дозволяє керувати виробничими процесами, планувати та розраховувати. Довідникова – технічну та технологічну документацію по спектру виробів, державні та міжнародні стандарти, документи що до використання та налагодження обладнання, таблиці витрат сировини, креслення та інше. Також довідникова частина містить у собі описи технологічних процесів. Невід'ємною частиною довідників є накопичуваний в виробництві досвід. Якщо у виробництві виникають будь-які складності або відхилення, можна продивитися як такі ситуації було вирішено в минулому.

Своєчасне планування допомагає оптимізувати виробничі процеси та зменшити витрати.

Основна частина коду - екранні форми та процедури для SQL.

Розглянемо роботу додатку. Початкове вікно після авторизації користувача це робочий стіл (Рис 3.8).

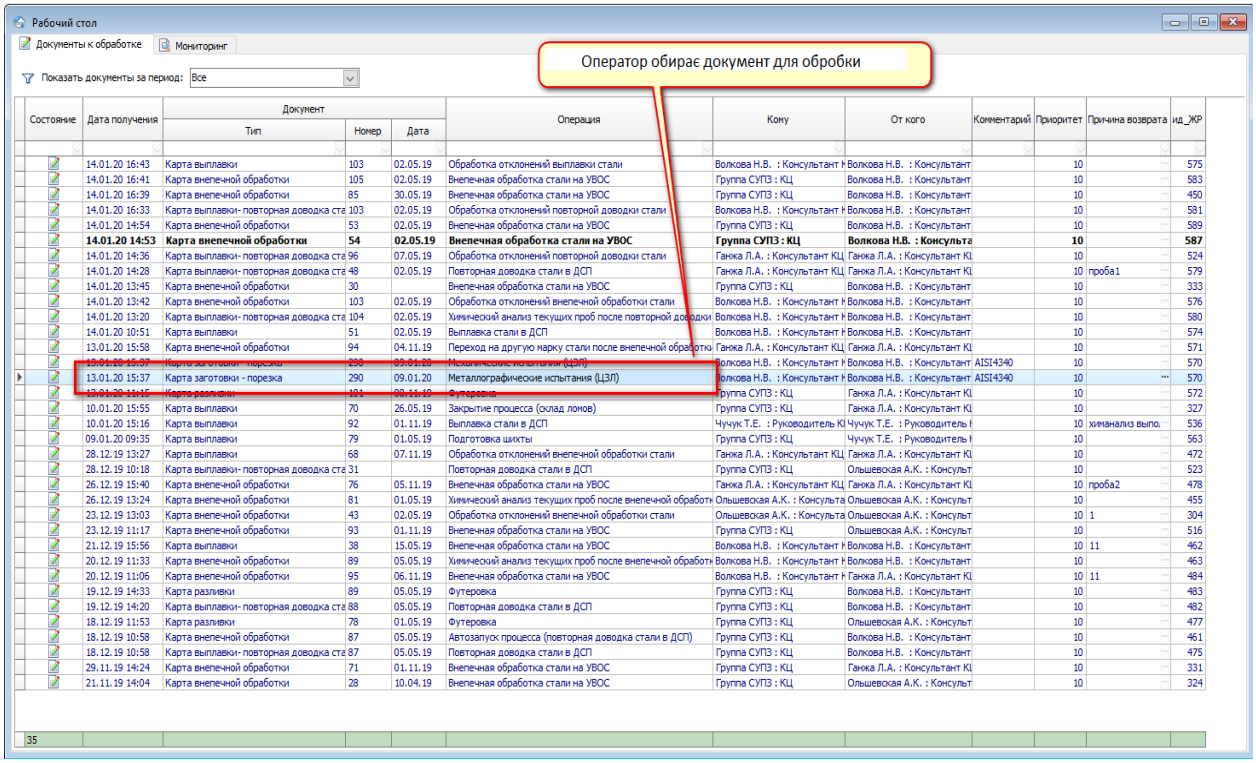


Рисунок 3.8 Робочий стіл.

Далі користувач бачить документ по обраному процесу, вигляд якого відповідає типу операції та регламентований СТП.

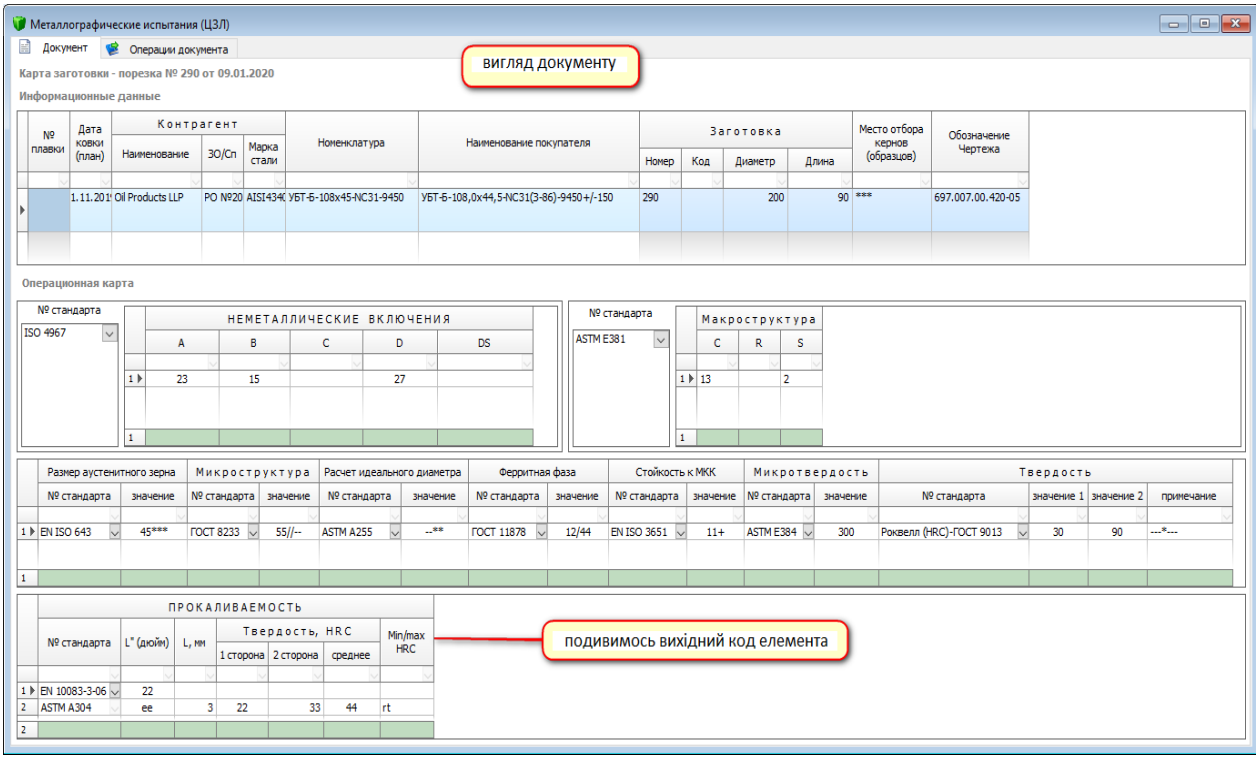


Рисунок 3.9 вигляд документа.

Подивимось як це виглядає з іншого боку.

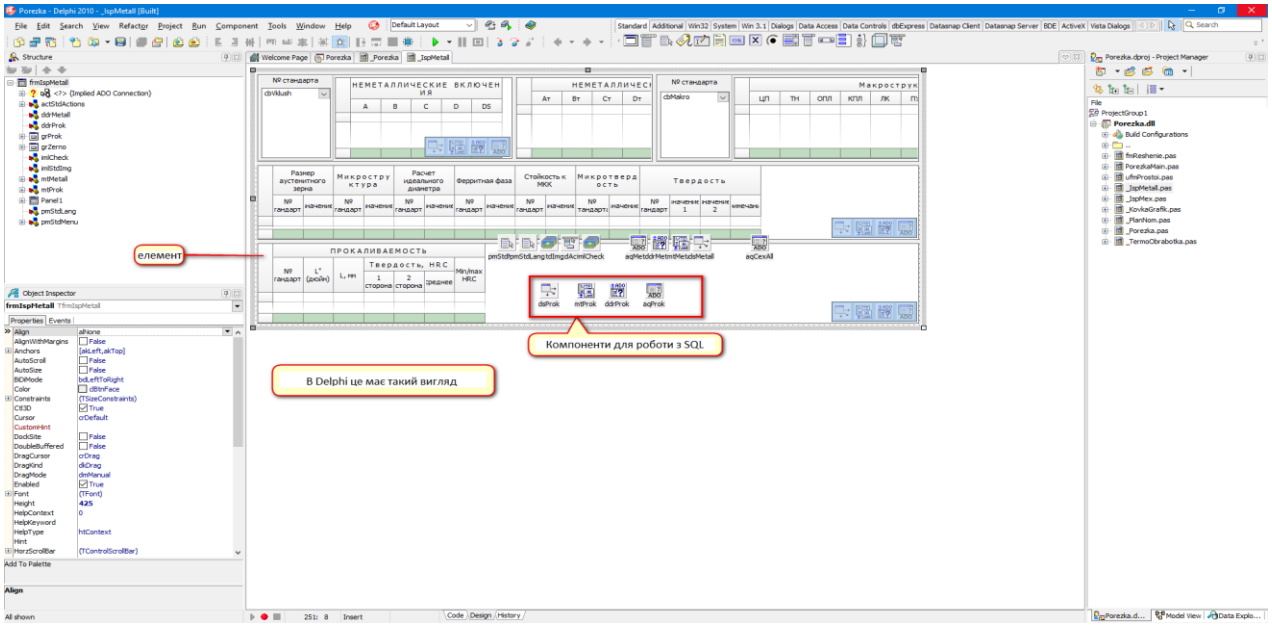


Рисунок 3.10 Вигляд з боку Delphi.

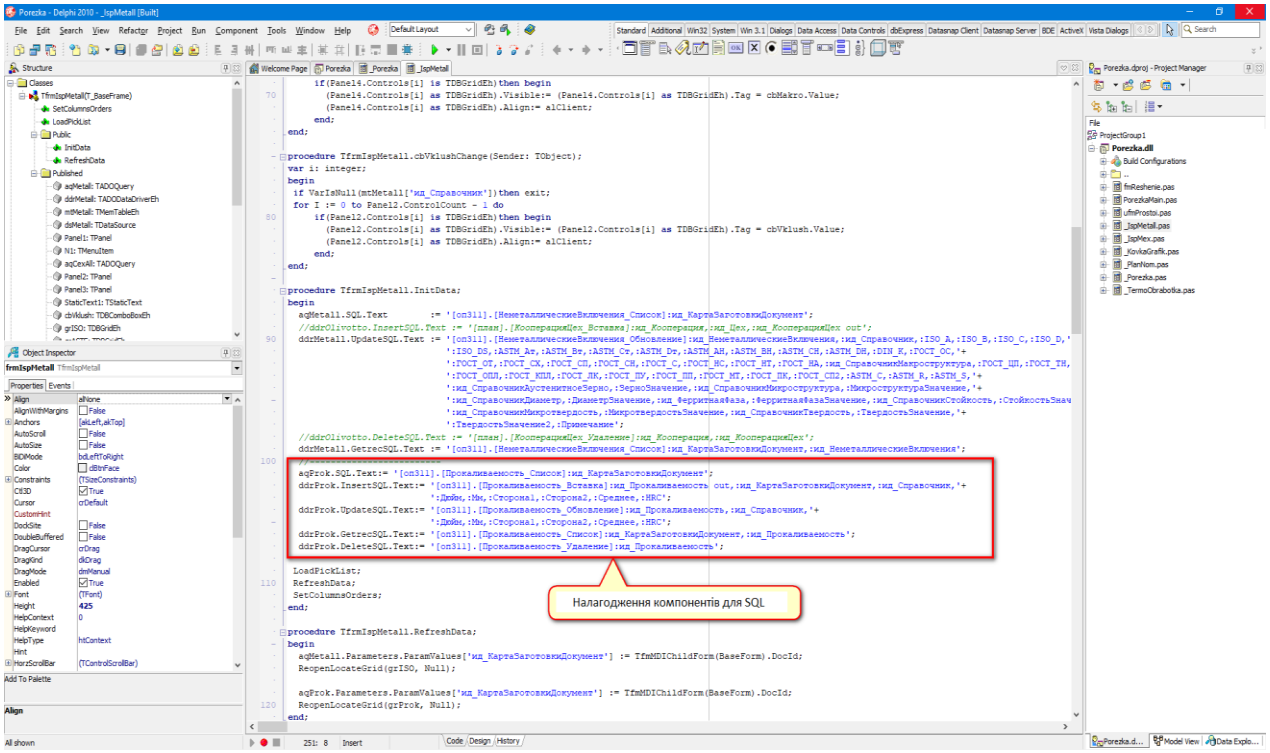


Рисунок 3.11 Налагодження компонентів для роботи з SQL

Розглянемо SQL процедуру «прокаливаемость_обновление» для SQL (див. рис. 3.12)

За допомогою команди USE обираємо робочу базу даних, в цьому випадку «xPlan».

USE [xPlan]

GO

*/****** Object: StoredProcedure [on311].[Прокаливаемость_Обновление] Script Date: 14.01.2020 20:38:41 ******/*

Для уникнення конфліктів з наступними версіями SQL Server використаємо команду

SET ANSI_NULLS ON

GO

Оскільки потрібна сумісність з даними XML, згідно з документацією, треба вказати

SET QUOTED_IDENTIFIER ON

GO

Ця команда змушує SQL Server дотримуватися правил ISO щодо лапок, що розмежовують ідентифікатори та буквальні рядки.

Процедура з таким ім’ям вже була присутня, тож її потрібно оновити:

ALTER PROCEDURE [on311].[Прокаливаемость_Обновление](--

Вказуємо тип даних в полях (int – integer – числове значення, varchar – Variable-size string data – текстові дані, в лапках пишеться формат числа або довжина строки).

```
@ид_Прокаливаемость int
  ,@ид_Справочник      int
  ,@Дюйм               varchar(20)
  ,@Мм                       numeric(4,2)
  ,@Сторона1           int
  ,@Сторона2           int
  ,@Среднее            int
  ,@HRC                varchar(10)
)
as
begin
```

В цьому проекті ми не використовуємо .NET SQLDataAdapter, тож для оптимізації потоку даних до клієнта вставляємо команду:

```
set nocount on
```

Це також вимкне відправлення з боку серверу повідомлень типу «nn rows affected», які можуть викликати помилки в деяких ORM.

Для обробки можливих помилок використано конструкцію TRY – CATCH.

Блок TRY обробляє інструкції в середині, та якщо виникають помилки, вони обробляються блоком CATCH.

```
begin try
        --
        UPDATE on311.Прокаливаемость
        set
                ид_Справочник   = @ид_Справочник
                ,Дюйм                   = @Дюйм
                ,Мм                     = @Мм

                ,Сторона1       = @Сторона1
                ,Сторона2       = @Сторона2
                ,Среднее        = @Среднее
                ,HRC            = @HRC

                ,сис_Изменил    = SUSER_SNAME()
                ,сис_ДатаИзменения  = SYSDATETIME()


        where
                ид_Прокаливаемость = @ид_Прокаливаемость
```

end try
begin catch

За допомогою команди exec викликаємо сценарій для обробки помилки.

exec ядро.сис_Ошибка_Обработать
end catch
end

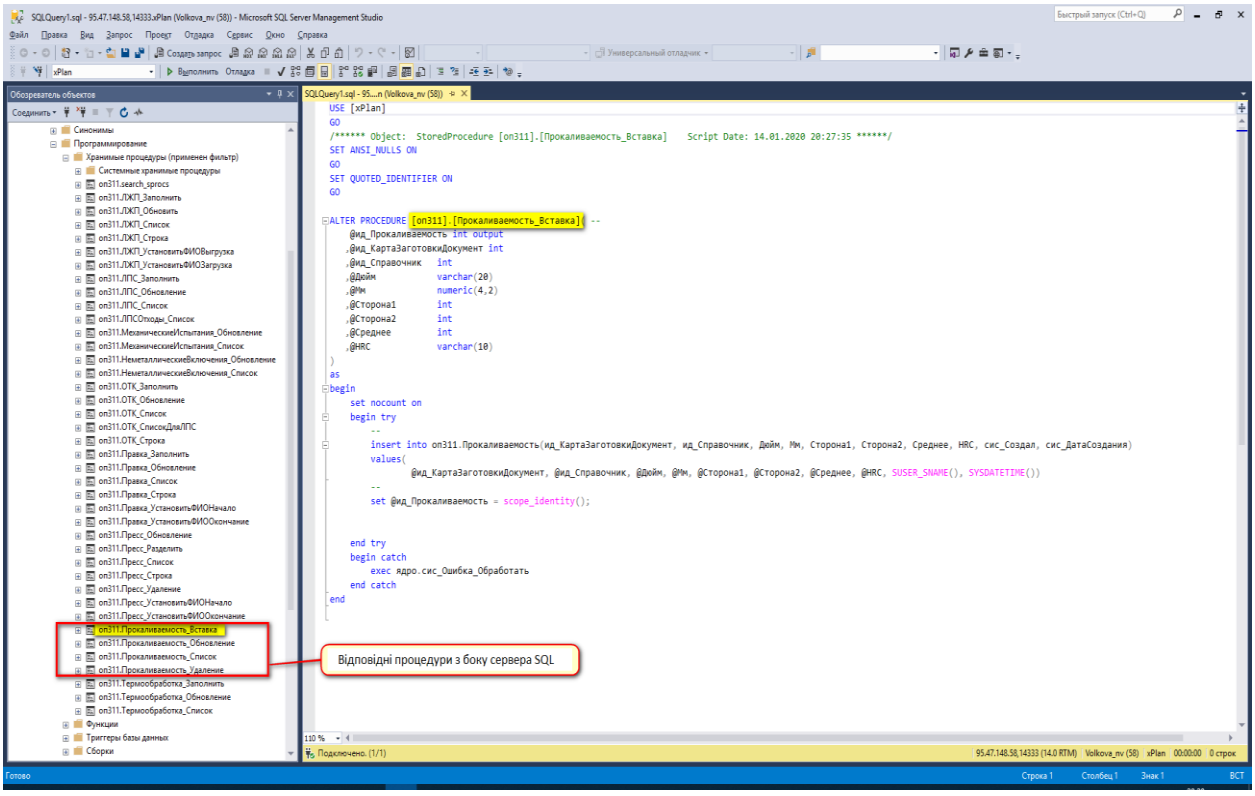


Рисунок 3.12 Процедури з боку SQL сервера.

ВИСНОВКИ

В процесі дослідження по створенню інформаційної системи автоматизації виробництва було:

- проведено аналітичний огляд існуючих інформаційних систем автоматизації виробництва;
- розроблено інформаційну систему автоматизації виробництва бурильних труб;
- обрано середовище програмування та інструменти для розробки інформаційної системи автоматизації виробництва бурильних труб;
- програмно реалізовано інформаційну систему автоматизації виробництва бурильних труб;
- протестовано інформаційну систему автоматизації виробництва бурильних труб.

ПЕРЕЛІК ЛІТЕРАТУРИ

1. Honestoblogz - http://honestoblogz.blogspot.com/2010/06/enterprise-resource-planning.html

2. Ситник В. Ф. Основи інформаційних систем. / Навчальний посібник Київ КНЕУ 2001. – 420 с.

3. Michael Tuck - https://www.sitepoint.com/real-history-gui/

4. Armstrong - «The Quarks of Object-Oriented Development.» https://cacm.acm.org/magazines/2006/2/5991-the-quarks-of-object-oriented-development/fulltext

5. The Gartner Glossary of Information Technology Acronyms and Terms Gartner (2004). – P. 390 - 395.

6. C. J. Date Introduction to Database Systems, 6th-ed., Page 650 (англ.)

7. В. М. Заяць - Функційне програмування : Навч. посіб. для студ. вищих навч. закл., що навч. за спец. "Програм. забезп. автоматиз. систем" Нац. ун-т "Львів. політехніка". - Л. : Бескид Біт, 2003. с.159

8.Infowatch - https://www.infowatch.ru/resources/analytics/reports/report2016-half

9. ERPNext - https://erpnext.com/

10. Actindo - https://www.actindo.com/en/

11 1С - http://1c.ua/ua/v8/

12. Amy Brown and Greg Wilson The Architecture of Open Source Applications – 2016. – 121с.

13. Robert W. Sebesta: Concepts of Programming Languages p. 43 – 46. https://github.com/winstondu/Algorithm_Papers/blob/master/Robert%20W.%20Sebesta%20Concepts%20of%20Programming%20Languages.pdf

14. IEEE documents - https://www.ieee.org/

15. David Intersimone. Borland History: Why the name «Delphi?» - https://web.archive.org/web/20100411053213/http:/edn.embarcadero.com/article/20396

16. Жовнірова М. В Удосконалення системи управління витратами на підприємствах. Науковий вісник НЛТУ України. – 2010. – № 19.2.

ДОДАТОК

Вихідний код форми:

```
unit _IspMetall;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, BaseFrame, MemTableDataEh, Db, ADODB, Menus, ActnList, ImgList,
  MemTableEh, DataDriverEh, ADODataDriverEh, StdCtrls, Buttons, DBCtrlsEh,
  ExtCtrls, GridsEh, DBGridEh, ComCtrls, ToolWin, Tabs, MDIChildForm, CommonUtils,
  DBGridEhGrouping, ToolCtrlsEh, DBGridEhToolCtrls, DynVarsEh, DBAxisGridsEh,
  EhLibVCL, Mask ;

type
  TfrmIspMetall = class(T_BaseFrame)
    aqMetall: TADOQuery;
    ddrMetall: TADODataDriverEh;
    mtMetall: TMemTableEh;
    dsMetall: TDataSource;
    Panel1: TPanel;
    N1: TMenuItem;
    aqCexAll: TADOQuery;
    Panel2: TPanel;
```

```
  Panel3: TPanel;
  StaticText1: TStaticText;
  cbVklush: TDBComboBoxEh;
  grISO: TDBGridEh;
  grASTE: TDBGridEh;
  grDIN: TDBGridEh;
  grPro: TDBGridEh;
  Panel4: TPanel;
  Panel5: TPanel;
  StaticText2: TStaticText;
  cbMakro: TDBComboBoxEh;
  grMakro: TDBGridEh;
  grMakro2: TDBGridEh;
  grZerno: TDBGridEh;
  grProk: TDBGridEh;
  dsProk: TDataSource;
  aqProk: TADOQuery;
  mtProk: TMemTableEh;
  ddrProk: TADODataDriverEh;
  procedure mtMetallBeforeDelete(DataSet: TDataSet);
  procedure mtMetallBeforeInsert(DataSet: TDataSet);
  procedure cbVklushChange(Sender: TObject);
  procedure cbMakroChange(Sender: TObject);
  procedure mtProkNewRecord(DataSet: TDataSet);
private
  procedure LoadPickList;
  procedure SetColumnsOrders;
public
   procedure RefreshData;
   procedure InitData;Override;
```

```
  end;

var IspMetall: TfrmIspMetall;

implementation

uses uStock;

{$R *.dfm}
//---------------------------------------------------------------------------
//   Выставить видимость сетки: Неметаллические включения
procedure TfrmIspMetall.cbMakroChange(Sender: TObject);
var i: integer;
begin
 if VarIsNull(mtMetall['ид_СправочникМакроструктура'])then exit;
 for I := 0 to Panel4.ControlCount - 1 do
    if(Panel4.Controls[i] is TDBGridEh)then begin
      (Panel4.Controls[i]  as  TDBGridEh).Visible:=  (Panel4.Controls[i]  as
TDBGridEh).Tag = cbMakro.Value;
      (Panel4.Controls[i] as TDBGridEh).Align:= alClient;
    end;
end;

procedure TfrmIspMetall.cbVklushChange(Sender: TObject);
var i: integer;
begin
 if VarIsNull(mtMetall['ид_Справочник'])then exit;
 for I := 0 to Panel2.ControlCount - 1 do
    if(Panel2.Controls[i] is TDBGridEh)then begin
```

```
            (Panel2.Controls[i] as TDBGridEh).Visible:= (Panel2.Controls[i] as
TDBGridEh).Tag = cbVklush.Value;
            (Panel2.Controls[i] as TDBGridEh).Align:= alClient;
          end;
        end;

        procedure TfrmIspMetall.InitData;
        begin
          aqMetall.SQL.Text                                              :=
'[оп311].[НеметаллическиеВключения_Список]:ид_КартаЗаготовкиДокумен
т';
          //ddrOlivotto.InsertSQL.Text                                   :=
'[план].[КооперацияЦех_Вставка]:ид_Кооперация,:ид_Цех,:ид_КооперацияЦ
ех out';
          ddrMetall.UpdateSQL.Text                                       :=
'[оп311].[НеметаллическиеВключения_Обновление]:ид_НеметаллическиеВкл
ючения,:ид_Справочник,:ISO_A,:ISO_B,:ISO_C,:ISO_D,'+

':ISO_DS,:ASTM_Am,:ASTM_Bm,:ASTM_Cm,:ASTM_Dm,:ASTM_AH,:ASTM_B
H,:ASTM_CH,:ASTM_DH,:DIN_K,:ГОСТ_ОС,'+

':ГОСТ_ОТ,:ГОСТ_СХ,:ГОСТ_СП,:ГОСТ_СН,:ГОСТ_С,:ГОСТ_НС,:ГОСТ_Н
Т,:ГОСТ_НА,:ид_СправочникМакроструктура,:ГОСТ_ЦП,:ГОСТ_ТН,'+

':ГОСТ_ОПЛ,:ГОСТ_КПЛ,:ГОСТ_ЛК,:ГОСТ_ПУ,:ГОСТ_ПП,:ГОСТ_МТ,:ГО
СТ_ПК,:ГОСТ_СП2,:ASTM_C,:ASTM_R,:ASTM_S,'+

':ид_СправочникАустенитноеЗерно,:ЗерноЗначение,:ид_СправочникМикрост
руктура,:МикроструктураЗначение,'+
```

```
':ид_СправочникДиаметр,:ДиаметрЗначение,:ид_ФерритнаяФаза,:ФерритнаяФазаЗначение,:ид_СправочникСтойкость,:СтойкостьЗначение,'+

':ид_СправочникМикротвердость,:МикротвердостьЗначение,:ид_СправочникТвердость,:ТвердостьЗначение,'+
                    ':ТвердостьЗначение2,:Примечание';
    //ddrOlivotto.DeleteSQL.Text :=
'[план].[КооперацияЦех_Удаление]:ид_Кооперация,:ид_КооперацияЦех';
    ddrMetall.GetrecSQL.Text :=
'[оп311].[НеметаллическиеВключения_Список]:ид_КартаЗаготовкиДокумент,:ид_НеметаллическиеВключения';
    //-----------------------
    aqProk.SQL.Text:=
'[оп311].[Прокаливаемость_Список]:ид_КартаЗаготовкиДокумент';
    ddrProk.InsertSQL.Text:=
'[оп311].[Прокаливаемость_Вставка]:ид_Прокаливаемость
out,:ид_КартаЗаготовкиДокумент,:ид_Справочник,'+
                    ':Дюйм,:Мм,:Сторона1,:Сторона2,:Среднее,:HRC';
    ddrProk.UpdateSQL.Text:=
'[оп311].[Прокаливаемость_Обновление]:ид_Прокаливаемость,:ид_Справочник,'+
                    ':Дюйм,:Мм,:Сторона1,:Сторона2,:Среднее,:HRC';
    ddrProk.GetrecSQL.Text:=
'[оп311].[Прокаливаемость_Список]:ид_КартаЗаготовкиДокумент,:ид_Прокаливаемость';
    ddrProk.DeleteSQL.Text:=
'[оп311].[Прокаливаемость_Удаление]:ид_Прокаливаемость';

    LoadPickList;
```

```
 RefreshData;
 SetColumnsOrders;
end;

procedure TfrmIspMetall.RefreshData;
begin
  aqMetall.Parameters.ParamValues['ид_КартаЗаготовкиДокумент']  :=
TfmMDIChildForm(BaseForm).DocId;
 ReopenLocateGrid(grISO, Null);

  aqProk.Parameters.ParamValues['ид_КартаЗаготовкиДокумент']    :=
TfmMDIChildForm(BaseForm).DocId;
 ReopenLocateGrid(grProk, Null);
end;

procedure TfrmIspMetall.SetColumnsOrders;
begin  (*
 with lbColumns.Items do begin  {$REGION ' Прочность '}
    clear;
    Add('ПределПрочностиМПа');
    Add('ПределПрочностиKsi');
    Add('ПределПрочностиPsi');
    Add('ПределТекучести2МПа');
    Add('ПределТекучести2Ksi');
    Add('ПределТекучести2Psi');
    Add('ПределТекучести5МПа');
    Add('ПределТекучести5Ksi');
    Add('ПределТекучести5Psi');
    Add('ПределТекучести6МПа');
    Add('ПределТекучести6Ksi');
```

```
Add('ПределТекучести6Psi');
Add('ПределТекучести1МПа');
Add('ПределТекучести1Ksi');
Add('ПределТекучести1Psi');
Add('ОтносительноеУдлинение');
Add('ОтносительноеСужение');
Add('ОтносительныйСдвиг');
Add('БоковоеУширение');
Add('Твердость3HB');
Add('Твердость3HRC');
Add('Твердость25HB');
Add('Твердость25HRC');
Add('ТвердостьQTCHB');
Add('ТвердостьQTCHRC');
Add('ид_Справочник');
Add('A1');
Add('A2');
Add('A3');
Add('A4');
Add('B1');
Add('B2');
Add('B3');
Add('B4');
Add('C1');
Add('C2');
Add('C3');
Add('C4');
Add('D1');
Add('D2');
Add('D3');
```

```
      Add('D4');
     {$ENDREGION}
   end;

   SetColumnsOrder(lbColumns, grPro);
   *)
  end;

  procedure TfrmIspMetall.LoadPickList;
  var ANameSpr: string;    AObject: TObject;
  begin
   aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список]';
    aqCexAll.Open;
    AObject:= cbVklush;
    FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

   aqCexAll.Close;
   aqCexAll.SQL.Text     :=     '[нси].[НеметаллическиеВключения_Список]
1275'; // Макроструктура
   aqCexAll.Open;
   AObject:= cbMakro;
   FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

   aqCexAll.Close;
   aqCexAll.SQL.Text     :=     '[нси].[НеметаллическиеВключения_Список]
1281'; // Размер аустенитного зерна
   aqCexAll.Open;
   AObject:= grZerno.FieldColumns['ид_СправочникАустенитноеЗерно'];
   FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');
```

```
aqCexAll.Close;
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1274'; // Микроструктура
aqCexAll.Open;
AObject:= grZerno.FieldColumns['ид_СправочникМикроструктура'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

aqCexAll.Close;
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1280'; // Расчет идеального диаметра
aqCexAll.Open;
AObject:= grZerno.FieldColumns['ид_СправочникДиаметр'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

aqCexAll.Close;
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1276'; // Ферритная фаза
aqCexAll.Open;
AObject:= grZerno.FieldColumns['ид_ФерритнаяФаза'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

aqCexAll.Close;
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1278'; // Стойкость к МКК
aqCexAll.Open;
AObject:= grZerno.FieldColumns['ид_СправочникСтойкость'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

aqCexAll.Close;
```

```
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1283'; // Микротвердость
aqCexAll.Open;
AObject:= grZerno.FieldColumns['ид_СправочникМикротвердость'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

aqCexAll.Close;
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1279'; // Твердость
aqCexAll.Open;
AObject:= grZerno.FieldColumns['ид_СправочникТвердость'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');

aqCexAll.Close;
aqCexAll.SQL.Text := '[нси].[НеметаллическиеВключения_Список] 1277'; // Прокаливаемость
aqCexAll.Open;
AObject:= grProk.FieldColumns['ид_Справочник'];
FillPickList(AObject, aqCexAll, 'ид_Справочник', 'Наименование');
end;

procedure TfrmIspMetall.mtMetallBeforeDelete(DataSet: TDataSet);
begin
 abort;
end;

procedure TfrmIspMetall.mtMetallBeforeInsert(DataSet: TDataSet);
begin
 abort;
end;
```

```
procedure TfrmIspMetall.mtProkNewRecord(DataSet: TDataSet);
begin
 mtProk['ид_КартаЗаготовкиДокумент']                    :=
TfmMDIChildForm(BaseForm).DocId;
end;

end.
```

Процедури для SQL:

1.

```
USE [xPlan]
GO
/****** Object:  StoredProcedure [on311].[Прокаливаемость_Вставка]
Script Date: 14.01.2020 20:27:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [on311].[Прокаливаемость_Вставка]( --
   @ид_Прокаливаемость int output
  ,@ид_КартаЗаготовкиДокумент int
  ,@ид_Справочник      int
  ,@Дюйм               varchar(20)
  ,@Мм                       numeric(4,2)
  ,@Сторона1           int
  ,@Сторона2           int
  ,@Среднее            int
  ,@HRC                varchar(10)
```

```
)
as
begin
	set nocount on
	begin try
		--
		insert into оп311.Прокаливаемость(ид_КартаЗаготовкиДокумент, ид_Справочник, Дюйм, Мм, Сторона1, Сторона2, Среднее, HRC, сис_Создал, сис_ДатаСоздания)
		values(
			@ид_КартаЗаготовкиДокумент, @ид_Справочник, @Дюйм, @Мм, @Сторона1, @Сторона2, @Среднее, @HRC, SUSER_SNAME(), SYSDATETIME())
		--
		set @ид_Прокаливаемость = scope_identity();


	end try
	begin catch
		exec ядро.сис_Ошибка_Обработать
	end catch
end
```

2.

```
USE [xPlan]
GO
/****** Object: StoredProcedure [оп311].[Прокаливаемость_Обновление] Script Date: 14.01.2020 20:38:41 ******/
SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [оп311].[Прокаливаемость_Обновление]( --
   @ид_Прокаливаемость int
  ,@ид_Справочник      int
  ,@Дюйм               varchar(20)
  ,@Мм                       numeric(4,2)
  ,@Сторона1           int
  ,@Сторона2           int
  ,@Среднее            int
  ,@HRC                varchar(10)
)
as
begin
        set nocount on
        begin try
                --
                UPDATE оп311.Прокаливаемость
                set
                                ид_Справочник   = @ид_Справочник
                               ,Дюйм                    = @Дюйм
                               ,Мм                      = @Мм

                               ,Сторона1        = @Сторона1
                               ,Сторона2        = @Сторона2
                               ,Среднее         = @Среднее
                               ,HRC             = @HRC
```

```
                    ,сис_Изменил        = SUSER_SNAME()
                    ,сис_ДатаИзменения    = SYSDATETIME()


                where
                    ид_Прокаливаемость = @ид_Прокаливаемость


            end try
            begin catch
                exec ядро.сис_Ошибка_Обработать
            end catch
        end
```

3.

```
USE [xPlan]
GO
/****** Object:  StoredProcedure [оп311].[Прокаливаемость_Список]    Script
Date: 14.01.2020 20:39:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [оп311].[Прокаливаемость_Список]( --
        @ид_КартаЗаготовкиДокумент    int
  ,@ид_Прокаливаемость int = null
)
as
begin
        set nocount on
```

```
	begin try
		--
		select	ид_Прокаливаемость
				,ид_КартаЗаготовкиДокумент
				,ид_Справочник
				,Дюйм
				,Мм
				,Сторона1
				,Сторона2
				,Среднее
				,HRC
				,сис_Создал
				,сис_ДатаСоздания
				,сис_Изменил
				,сис_ДатаИзменения
		FROM
				 оп311.Прокаливаемость
		where
				ид_КартаЗаготовкиДокумент = @ид_КартаЗаготовкиДокумент
				and
				ид_Прокаливаемость = isnull(@ид_Прокаливаемость, ид_Прокаливаемость)

	end try
	begin catch
		exec ядро.сис_Ошибка_Обработать
	end catch
end
```

4.

```
USE [xPlan]
GO
/****** Object:  StoredProcedure [оп311].[Прокаливаемость_Удаление]
Script Date: 14.01.2020 20:39:15 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [оп311].[Прокаливаемость_Удаление]( --
   @ид_Прокаливаемость int
)
as
begin
	set nocount on
	begin try
		--
		delete оп311.Прокаливаемость

		where
				ид_Прокаливаемость = @ид_Прокаливаемость


	end try
	begin catch
		exec ядро.сис_Ошибка_Обработать
	end catch
end
```