ДОДАТОК А

Програмний код

Вихідний код проекту

```
//// DDClassDictionary.h// testNewMailSender//// Created by Ermashov.. All rights reserved.//#import <Foundation/Foundation.h>/** Class dictionary of classes */@interface DDClassDictionary : NSObject/**get object(Class) fo key(Class) @param aKey The key(Class) according to which it will be possible to get the value(Class) @returns object(Class) corresponding to the key */- (Class)classForKey☹Class)aKey;/** set object(Class) fo key(Class) @param classObject stored object(Class) @param aKey The key(Class) according to which it will be possible to get the value(Class) */- (void)setClass☹Class)classObject forKey☹Class)aKey;@end//
```

```
 // DDClassDictionary.m// testNewMailSender//// Created by
Ermashov.. All rights reserved.//#import
«DDClassDictionary.h»@interface DDClassDictionary()@property
(nonatomic,strong) NSMutableDictionary
*clssDictionary;@end@implementation DDClassDictionary@synthesize
clssDictionary;- (id)init{ self = [super init]; if (self)
{  self.clssDictionary = [ [NSMutableDictionary alloc] init]; }
return self;}- (Class)classForKey☹Class)aKey{ NSString *className =
[clssDictionary valueForKey:NSStringFromClass (aKey)]; Class class =
NSClassFromString(className);
```

```
  return class;}- (void)setClass☹Class)classObject
forKey☹Class)aKey{ NSString *classString = NSStringFromClass
(classObject); NSString *keyString = NSStringFromClass (aKey);
[clssDictionary setValue:classString forKey:keyString];}@end////
DDCoreDataOperation.h// testDataManager//// Created by Bondarenko
Alexander on 1/17/13.. All rights reserved.//#import
<Foundation/Foundation.h>#import <CoreData/CoreData.h>/**  class
operation creates a CoreData context when performing */@interface
DDCoreDataOperation : NSOperation/**   initializes the object,
recommended to call this method during initialization @param context
CoreData main context in which the merger will be@param block block
that is called when the operation is starting performed    @returns
return self */-
(id)initWithMainManagedObjectContext☹NSManagedObjectContext
*)context withBlock☹void(^)(NSManagedObjectContext
*context))block;@end//// DDCoreDataOperation.m// testDataManager////
Created by Bondarenko Alexander on 1/17/13.. All rights
reserved.//#import «DDCoreDataOperation.h»@interface
NSManagedObjectContext (Additions)- (BOOL) save;@end@implementation
NSManagedObjectContext (Additions)- (BOOL) save{ NSError *error =
nil; BOOL status = YES; if ( [self hasChanges]) {  if (! [self
save:&error])  {   NSLog(@»Unresolved context save error: %@, %@»,
error, [error userInfo]);
```

```
    status = NO;   } } return status;}@end@interface
DDCoreDataOperation(){ void(^block)(NSManagedObjectContext*
context);  NSManagedObjectContext *mainManagedObjectContext_;
NSManagedObjectContext *operationManagedObjectContext_;
NSMutableArray *changeNotifications_;}@property (nonatomic, assign)
NSThread *initialThread;@property (nonatomic, strong)
NSManagedObjectContext *mainManagedObjectContext;@property
(nonatomic, strong) NSManagedObjectContext
*operationManagedObjectContext;@property (nonatomic, strong)
NSMutableArray *changeNotifications;- (void)
processMainContextChangeNotification☹NSNotification*)notification;-
(void) mergeMainContextChangesIntoOperationContext;- (void)
processOperationContextChangeNotification☹NSNotification*)notificati
on;- (void)
mergeOperationContextChangesIntoMainContext☹NSNotification*)notifica
tion;- (void) doMain;@end
```

```
 @implementation DDCoreDataOperation@synthesize
mainManagedObjectContext = mainManagedObjectContext_;@synthesize
operationManagedObjectContext =
operationManagedObjectContext_;@synthesize changeNotifications =
changeNotifications_;- (void)
processMainContextChangeNotification☹NSNotification*)notification{ @
synchronized(self) {   [self.changeNotifications
addObject:notification]; }}- (void)
mergeMainContextChangesIntoOperationContext{ @synchronized(self)
{  if ( [changeNotifications_ count] > 0)  {   for(NSNotification*
change in self.changeNotifications)
{     [operationManagedObjectContext_
mergeChangesFromContextDidSaveNotification:change];   }
[operationManagedObjectContext_ save];    [self.changeNotifications
removeAllObjects];  } }}- (void)
```

```
 processOperationContextChangeNotification☹NSNotification*)notificat
ion{ @synchronized(self) {   [self.changeNotifications
addObject:notification]; }   [self
performSelector:@selector(mergeOperationContextChangesIntoMainContex
t☺     onThread:self.initialThread    withObject:notification
waitUntilDone:NO];}- (void)
mergeOperationContextChangesIntoMainContext☹NSNotification*)notifica
tion{  [mainManagedObjectContext_
mergeChangesFromContextDidSaveNotification:notification];
[mainManagedObjectContext_ save];}- (id)
initWithMainManagedObjectContext☹NSManagedObjectContext*)context
withBlock☹void(^)(NSManagedObjectContext* context))newBlock;{ if
((self = [super init])) {  block = newBlock;  self.initialThread =
[NSThread currentThread];  self.mainManagedObjectContext = context;
self.operationManagedObjectContext = nil;  self.changeNotifications
= [NSMutableArray arrayWithCapacity:4];     [ [NSNotificationCenter
defaultCenter] addObserver:self
selector:@selector(processMainContextChangeNotification☺
name:NSManagedObjectContextDidSaveNotification
object:mainManagedObjectContext_];   } return self;}- (void)
main{ @autoreleasepool {  @try  {   if (! [self isCancelled])   {
self.operationManagedObjectContext = [ [NSManagedObjectContext
alloc] init];     [operationManagedObjectContext_
setPersistentStoreCoordinator: [mainManagedObjectContext_
persistentStoreCoordinator]];         [ [NSNotificationCenter
defaultCenter] addObserver:self
selector:@selector(processOperationContextChangeNotification☺
name:NSManagedObjectContextDidSaveNotification
object:operationManagedObjectContext_];         [self doMain];
[self mergeMainContextChangesIntoOperationContext];   }
```

```
   }  @catch(NSException* exception)  {   NSLog(@»Exception: %@,
%@», exception.name, exception.reason);  }  @catch(...)
{   NSLog(@»%@», @»Not NSException-based exception!»);  }  @finally
{    [ [NSNotificationCenter defaultCenter] removeObserver:self];
self.operationManagedObjectContext = nil;  }   }}- (void) doMain{ if
(block) {  block(self.operationManagedObjectContext);   }}#pragma
mark Memory management- (void) dealloc{ self.initialThread = nil;
self.mainManagedObjectContext = nil; self.changeNotifications =
nil;}
```

ДОДАТОК Б

Слайди презентації

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Атестаційна робота магістра

ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБКИ ЕНЕРГОЗБЕРІГАЮЧОГО ПЗ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

Керівник:

проф. Шостак І.В.

Виконав:

ст.гр. ІПЗм-18-4
Дригулич Є.С.

2020

1

Мета роботи

розробка алгоритмів продовження тривалості роботи мобільного пристрою, без підзарядки від джерела живлення, **за рахунок оптимізації алгоритмів роботи ПЗ**, за критерієм мінімуму енергоспоживання

Підхід, який шляхом незначної оптимізації логічних процесів, що відбуваються при виконанні ПЗ, відчутно збільшить час роботи акумулятора, а так же, з причини своєї специфіки, знизить час завантаженості відеокарти і процесора, що в свою чергу призведе до зменшення нагрівання даних компонентів.

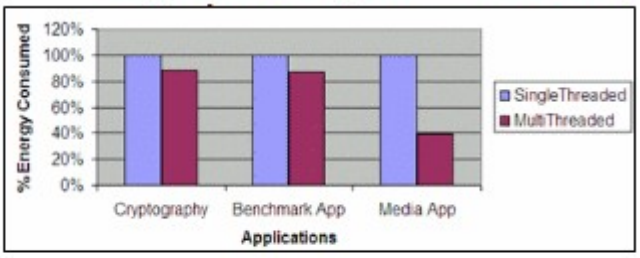
Даний підхід відповідає таким принципам:

- використовувати ресурси пристрою настільки, наскільки потрібно (немає сенсу проводити розрахунки в одному потоці, якщо пристрій має, наприклад, 4-х ядерним процесором);
- діяти в рамках контексту (працювати в різних режимах в залежності від того, що зараз є джерелом живлення: мережа або акумулятор).

Комбінація двох цих методів (поліпшення апаратної складової і оптимізація ПЗ) дозволить досягти збільшення терміну роботи пристрою без доступу до електромережі.

2

Економія електроенергії трьох різних додатків в однопоточному і в багатопотоковому



Приклад оптимізації витрати електрики системою зберігання даних

Access Interval	100 mS	1S	5S	10 S	20S
SSD Average Power	0.5W	208mW	154mW	89mW*	
HDD Average Power	1.4 W	1.1 W	1.02 W	1.14 W	0.98 W

3

Аналіз способів зниження енергосповживання

Доведена необхідність впровадження стратегій оптимізації мобільного ПЗ.

Так само розглянуті види мобільних платформ і особливості оптимізації ПЗ для планшетів і смартфонів.

Проведений аналіз існуючих способів зниження енергосповживання за рахунок підвищення ефективності ПЗ і прикладів стратегій, **довів необхідність застосування комплексних підходів щодо оптимізації енергоефективності ПЗ мобільних пристроїв**, тим самим підвищуючи свою ефективность.

4

Завдання

- виконати аналіз проблеми енергоспоживання мобільних пристроїв при існуючих підходах до розробки ПЗ;
- виконати аналіз можливості впровадження нових підходів до розробки енергоефективного додатку;
- розробити загальні стратегії для різних типів мобільних пристроїв і платформ;
- сформулювати вимоги і вихідні дані для прототипу екологічно чистого додатку;
- розробити прототип додатка для експеременту;
- виконати верифікацію прототипу ПЗ;
- оптимізувати алгоритми роботи програми прототипу;
- виконати порівняльну оцінку внесених оптимізацій і визначити їх доцільність.

5

Основні вимоги до оптимізації енергоспоживання додатків

- Найбільш очевидними видами оптимізації є виключення відтворення фонової музики, при згортанні додатка, а так само відключення відтворення.
- По можливості варто поставити гру на паузу і зупинити всі внутрішньоігрові процеси, але тут все залежить від характеру процесів і жанру гри.
- Повідомлення про зменшення запасу батареї і запиту на зменшення яскравості, не обтяжить користувача, а дасть йому можливість самому вирішувати, що для нього зараз пріоритетне.
- З більш радикальних оптимізацій є зниження якості ігрових текстур, частоти перемальовування та ін., Але такі дії варто робити тільки після запиту на дозвіл у користувача

6

Основні принципи оптимізації по енергоспоживанню це:

- обчислювати швидко;
- використовувати ресурси пристрою настільки, наскільки потрібно (немає сенсу проводити розрахунки в одному потоці, якщо пристрій має до прикладу 4-х ядерним процесором);
- діяти в рамках контексту (працювати в різних режимах в залежності від того, що зараз є джерелом живлення: мережа або акумулятор).

7

Стратегії і обрані найоптимальніші методи

для зниження витрат заряду акумулятора, такі критерії:

- результат (процентна характеристика відображає зменшення енергоспоживання);
- простота впровадження (умовна характеристика відображає швидкість внесення змін до проекту);
- вплив на загальне враження про програму (якщо для даної оптимізації потрібно постійно вимкнений дисплей – цим додатком ніхто не буде користуватися);
- вплив на якість і загальну швидкість роботи програми (якщо при впровадженні оптимізації постраждає якийсь інший компонент або загальна швидкість роботи, можливо варто відмовитися від даної оптимізації).

8

Критерій «Економія енергії»

вимірюється в процентах – результат застосування стратегії до конкретного програмного продукту, різниця між споживаної раніше і споживаної зараз енергією.

$$S = \int_{t_1}^{t_2} f(t)dx$$

має сенс в якості параметрів для порівняння використовувати площу фігури відсіченою відрізками і кривої енергоспоживання

9

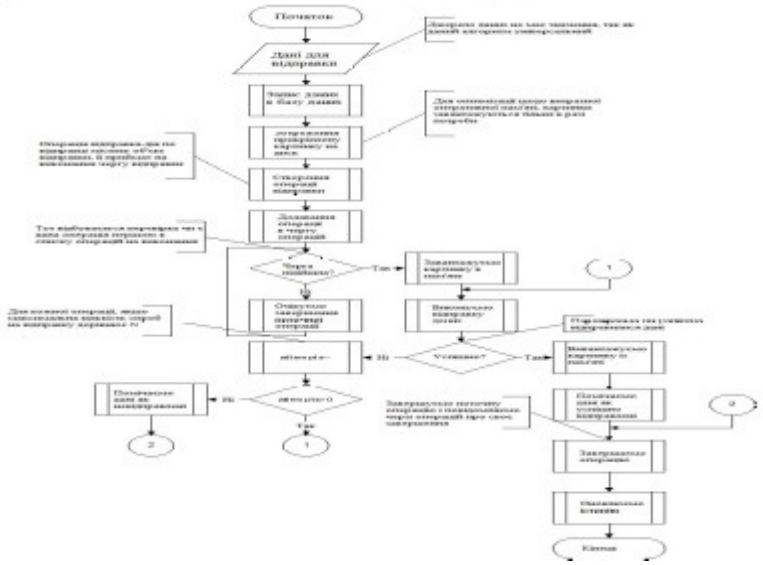
Оцінка ефективності покрашень

Вироблені критерії і обмеження для об'єктивної оцінки:

- економія енергії (критерій), має високе значення, так як є кінцевим результатом внесення оптимізації;
- трудовитрати (обмеження), мають найвищий пріоритет, бо важливо, в першу чергу, закінчити програмний продукт раніше конкурентів або укластися в терміни;
- глобальність змін (обмеження), від глобальності змін залежить час, який доведеться витратити на тестування;
- зручність для користувача інтерфейсу (обмеження), дуже важливий параметр, так як будь-який програмний продукт розробляється для кінцевого користувача.

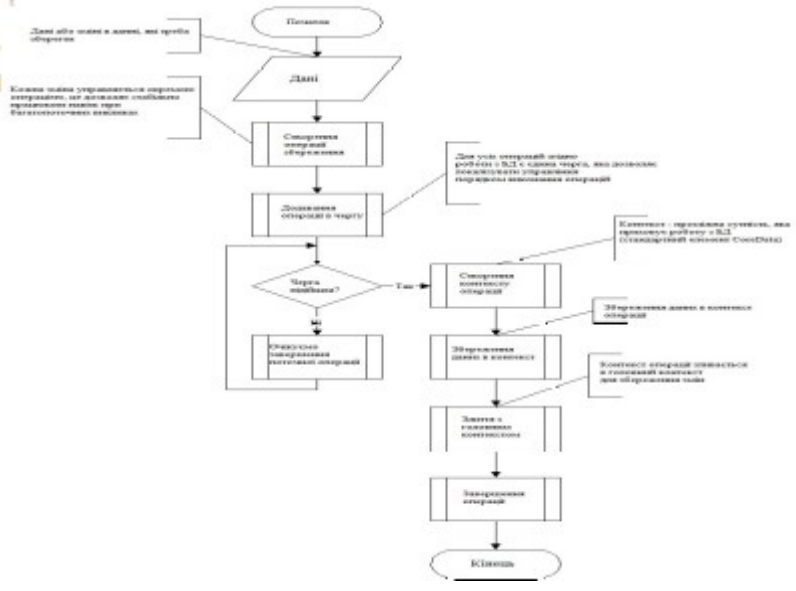
10

Алгоритм додавання даних на відправку в чергу



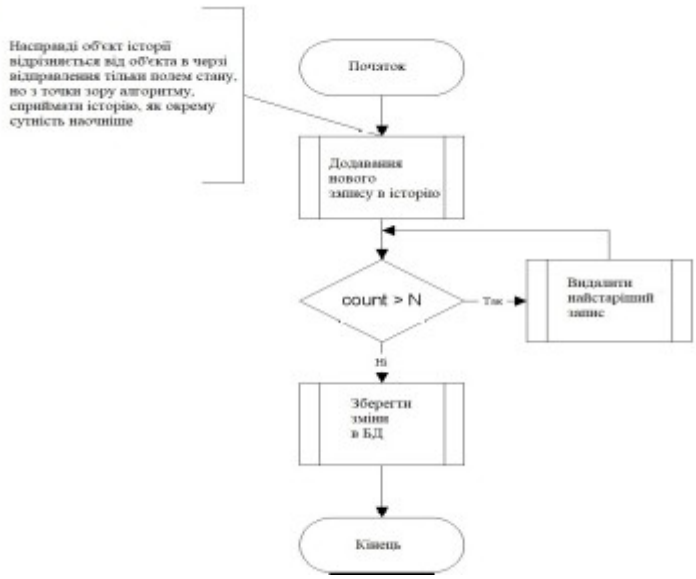
11

Алгоритм збереження в базу даних



12

Алгоритм поновлення історії відправок



13

Критерії вибору засобів розробки

При створенні програмного продукту основними критеріями вибору засобів розробки були:

- зручність використання;
- швидкість розробки додатків і роботи програми.

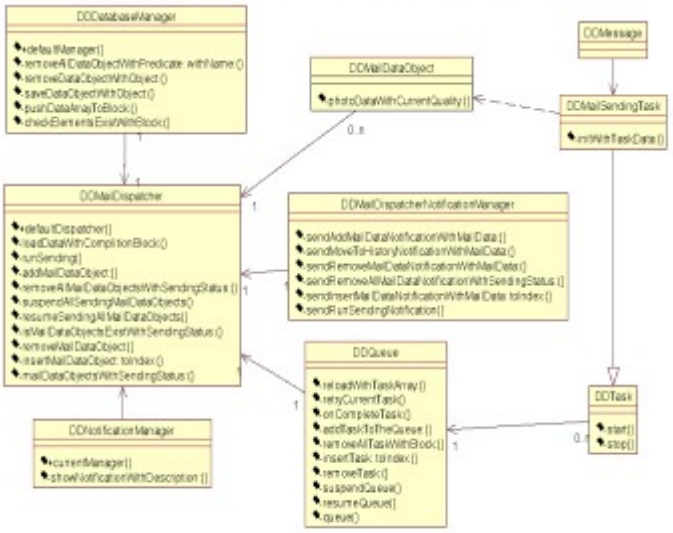
14

Архітектура ПЗ, що розробляється

На даний момент вся робота з відправкою прихована за класом **`DDMailDispatcher`**, він дає дані внутрішньої черги повідомлень, відправляє на збереження дані в **`DDDatabaseManager`**, а також відправляє повідомлення про стані відправки в **`DDMailDispatcherNotificationManager`**, для їх подальшої розсилці по класах з необхідною інформацією

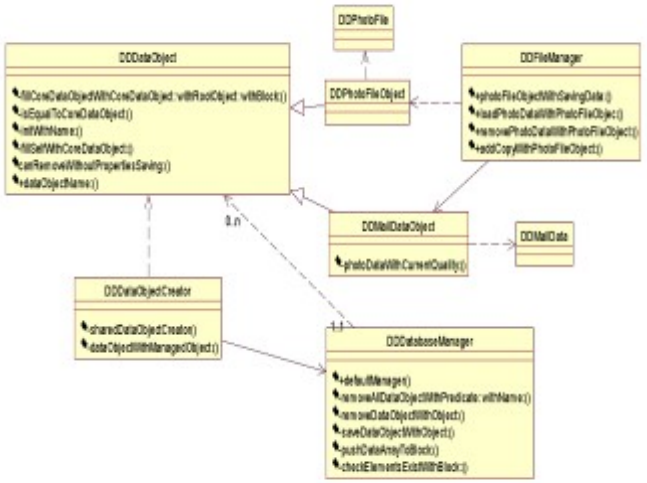
15

Діаграма класів (`DDMailDispatcher` і базові компоненти відправки)



16

Діаграма класів DDDatabaseManager і основні модельні дані



17

Графік енергоспоживання пристрою без запущених додатків

18

Графік енергоспоживання пристрою з активним застосуванням DDMailSender

19

Графік енергоспоживання пристрою з активним застосуванням DDMailSender (збільшено кількість потоків, що беруть участь у відправці)

20

Вибір оптимізації методом варіантних мереж

	Трудовитрати (4)	Економія енергії (2)	Глобальність змін (1)	Зручність для користувача інтерфейсу (3)	Разом
1	5	5	5	5	50
2	4	3	4	1	29
3	3	4	3	5	38

За результатами вимірювань, найбільшу доцільність впровадження має перша оптимізація, але з точки зору універсальності і стабільності роботи уваги заслуговує **третя оптимізація**, бо вона позитивно позначається на завантаженні оперативної пам'яті і дозволяє працювати з великими даними не завантажуючи їх повністю в пам'ять. Друга оптимізація є не доцільною, так як погіршує загальну роботу програми.

21

Висновки

Виділені ключові аспекти, на які треба звертати увагу при розробці екологічного ПЗ.

Були виділені критерії, за якими буде відбуватися оцінка внесених оптимізацій в програмного продукту:

Розроблено і описано основні алгоритми, які використовувалися при розробці прототипу додатка для перевірки стратегій оптимізації програмного продукту.

Результати мають такий вигляд:

- збільшення кількості потоків, що беруть участь у відправці – 38% економії енергії при мінімальних витратах на впровадження;
- мінімізація дискових операцій – 9% економії енергії, погіршення роботи програми та зниження його стабільності;
- потокове читання файлів з диска в процесі відправки – 17% економії енергії, підвищення стабільності додатків.

Розроблені алгоритми оптимізації ПЗ дозволяють скоротити енергоспоживання мобільного пристрою.

22

ДОДАТОК В

Апробація результатів роботи

Подано тези доповіді на Міжнародний молодіжний форум «Радіоелектроніка і молодь в XXI сторіччі

ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБКИ ЕНЕРГОЗБЕРІГАЮЧОГО ПЗ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

Дригулич Є.С.
Науковий керівник – проф. Шостак І.В.
Харківський національний університет радіоелектроніки
(61166, Харків, пр. Науки, 14, каф. Програмної інженерії,
тел.: (057) 702-14-46)
E-mail: i.v.shostak@gmail.com

In the work the method of estimating the maturity (perfection) of the projects performed in the testing of software systems is developed. This method is based on TMM's five-level maturity model. Also, the use of mathematical model, the basic process of testing software data processing systems under the conditions of limited resources, as well as several methods aimed at improving the quality of software products, have been developed and substantiated.

До головних напрямків програмної інженерії відносяться задачі вдосконалення процесів життєвого циклу ПС, зокрема процесу тестування. Неухильне підвищення якості програмних продуктів – основна мета програмної інженерії та предмет турботи розробників ПС.

Відповідь на питання, як підвищити конкурентоспроможність українських програмних продуктів, як знизити ризик проектів, як досягти балансу сторін трикутника «тривалість – вартість – цілі проекту», протягом останніх років намагаються знайти відповідь не лише керівники організацій-розробників ПЗ і менеджери проектів, але і замовники, і споживачі, що використовують програмні продукти низької якості. Наслідки постачання програмних продуктів низької якості – це завжди збитки користувачів: не лише матеріальні і фінансові втрати, але і падіння престижу. Ці проблеми, в свою чергу, негативно відображаються на конкурентоспроможності організації-розробника програмних продуктів. Для забезпечення необхідного рівня якості ПС в міжнародній практиці знаходять застосування два підходи: продукто-орієнтований і процесо-орієнтований. В першому акцент робиться на оцінку якості шляхом тестування готового програмного продукту. Цей підхід базується на припущенні, що чим більше знайдено і усунено дефектів в ПС при тестуванні, тим вище його якість.

Тестування є важливий етап розроблення ПС, оскільки, з одного боку, вимагає значних витрат на проведення, а з іншого – робить великий внесок у їхню якість. Через теоретично доведену неможливість вичерпного тестування та велику потенційну вартість втрат через відмови у ПС, потрібен чітко визначений та ефективний *процес тестування*, базований на ухвалені збалансованих рішень щодо тривалості та вартості тестування для досягнення необхідного рівня довіри до якості ПС.

Методи визначення кількісних критеріїв завершення тестування та керування процесом тестування з використанням кількісних вимірів ще мало використовуються в проектах створення ПС, що призводить до того, що якість та надійність залишаються не передбачуваними. Лише за наявності достовірної та своєчасної інформації щодо стану ПС, видів ризиків і можливих втрат через відмови може бути забезпечене ефективне виконання процесу тестування.

Метою роботи є проведення комплексу досліджень з інженерії тестування ПС обробления даних, формування ефективної стратегії тестування програмних систем, спрямованої на зниження ризику відмов під час експлуатації. Для цього в роботі розв'язуються наступні задачі:

- дослідження сучасних вимог до процесу тестування ПС;
- аналіз існуючих моделей надійності ПС, розроблення алгоритмів та програм їх реалізації;
- побудова моделі визначення оптимального часу тестування модулів ПС з урахуванням ризиків відмов,
- визначення структури базового процесу, що регламентує всі дії з підготовки, проведення та оцінювання результатів тестування, та розроблення методики виконання процесу тестування ПС обробленні даних;
- проектування та реалізація програмного комплексу підтримки інженерії тестування;
- аналіз сучасних підходів до визначення рівня зрілості процесу тестування ПС та розроблення методики оцінювання процесу тестування;
- впровадження запропонованих підходів, моделей та методик інженерії тестування в проекти з розроблення ПС обробленні даних та опис результатів випробування запропонованих моделей оцінювання оптимального часу тестування та методу оцінювання ризиків відмов програмних модулів.

З метою визначення ефективності впровадження базового процесу тестування був розроблений метод оцінювання зрілості (досконалості) виконуваних у проектах дій з тестування ПС. Цей метод ґрунтується на п'ятирівневій моделі зрілості TMM.

Випробування моделей мають виконуватися в конкретній інформаційно-аналітичної системі підтримки прийняття управлінських рішень, яка має складатися з програмних комплексів, об'єднаних діловим процесом обробленні даних. В ході практичної реалізації аналіз загроз відмов системи показав, що з позицій цілісності інформації найбільший внесок у ризик її відмов робить ПК контролю і введення даних до БД ORACLE, який встановлюється та одночасно функціонує на 10 робочих місцях. Для п'яти модулів цього ПК були виконані оцінки ризику відмов та часу тестування.