Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота

з курсу

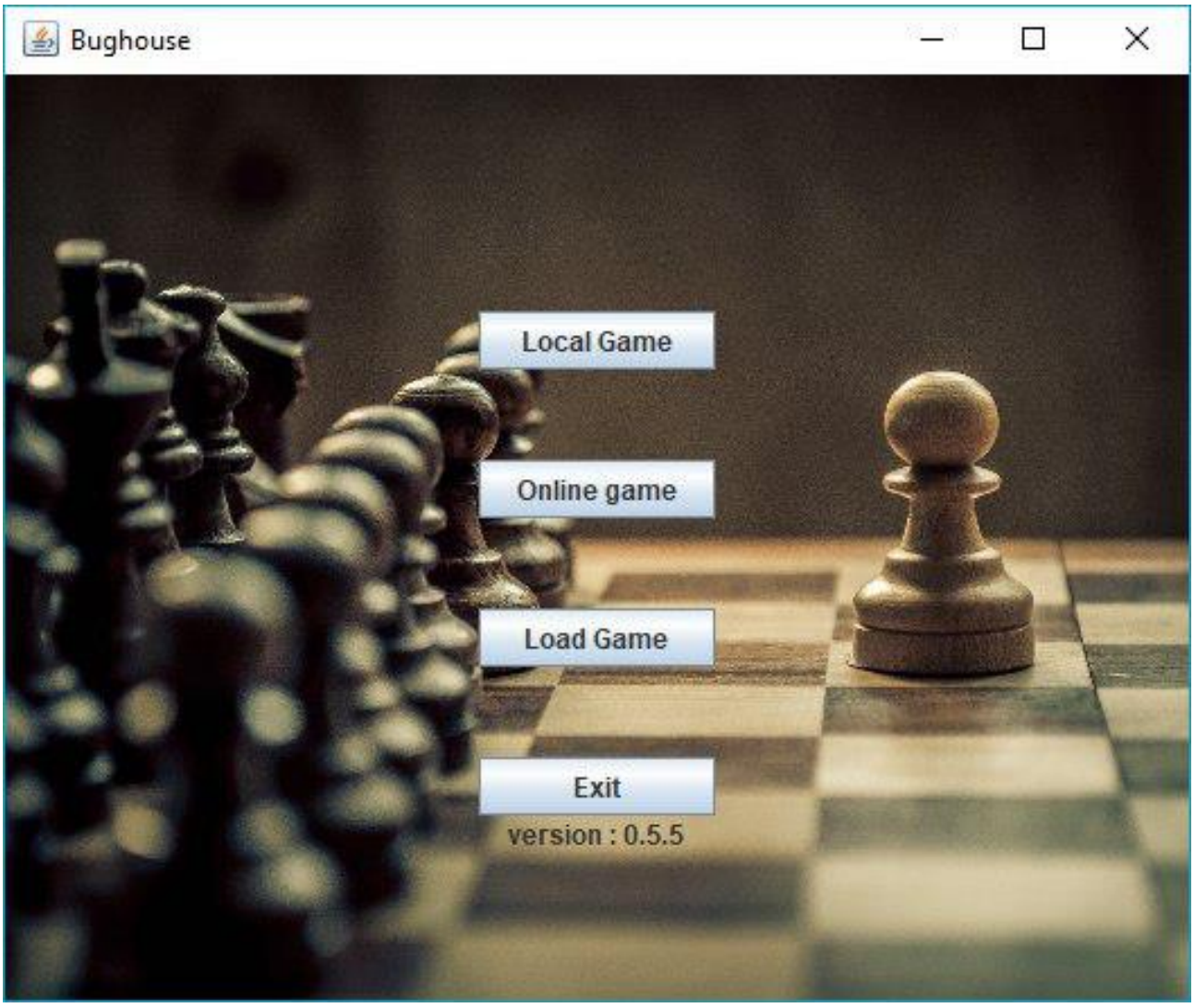
«Інтеграційні програмні системи»

Виконали:
студенти 4 курсу
ФІОТ гр. ІО-42
Слюсаренко О. Є.
Серпученко М. В.
Приправка Д. Г.
Красніков А. В.

Київ – 2017

1. Опис проекту

Розроблений нами проект дозволяє чотирьом користувачам грати в шведські шахи, як локально, на одному пристрої, так і через мережу інтернет. Зберігати гру та змінювати опції гри. Наша програма шведські шахи максимально наближена до гри у реальному житті те, наскільки ви граєте чесно та за правилами, залежить тільки від вас, так само як і у житті. Це робить нашу програму унікальною та особливою і відмінною від інших подібних додатків.

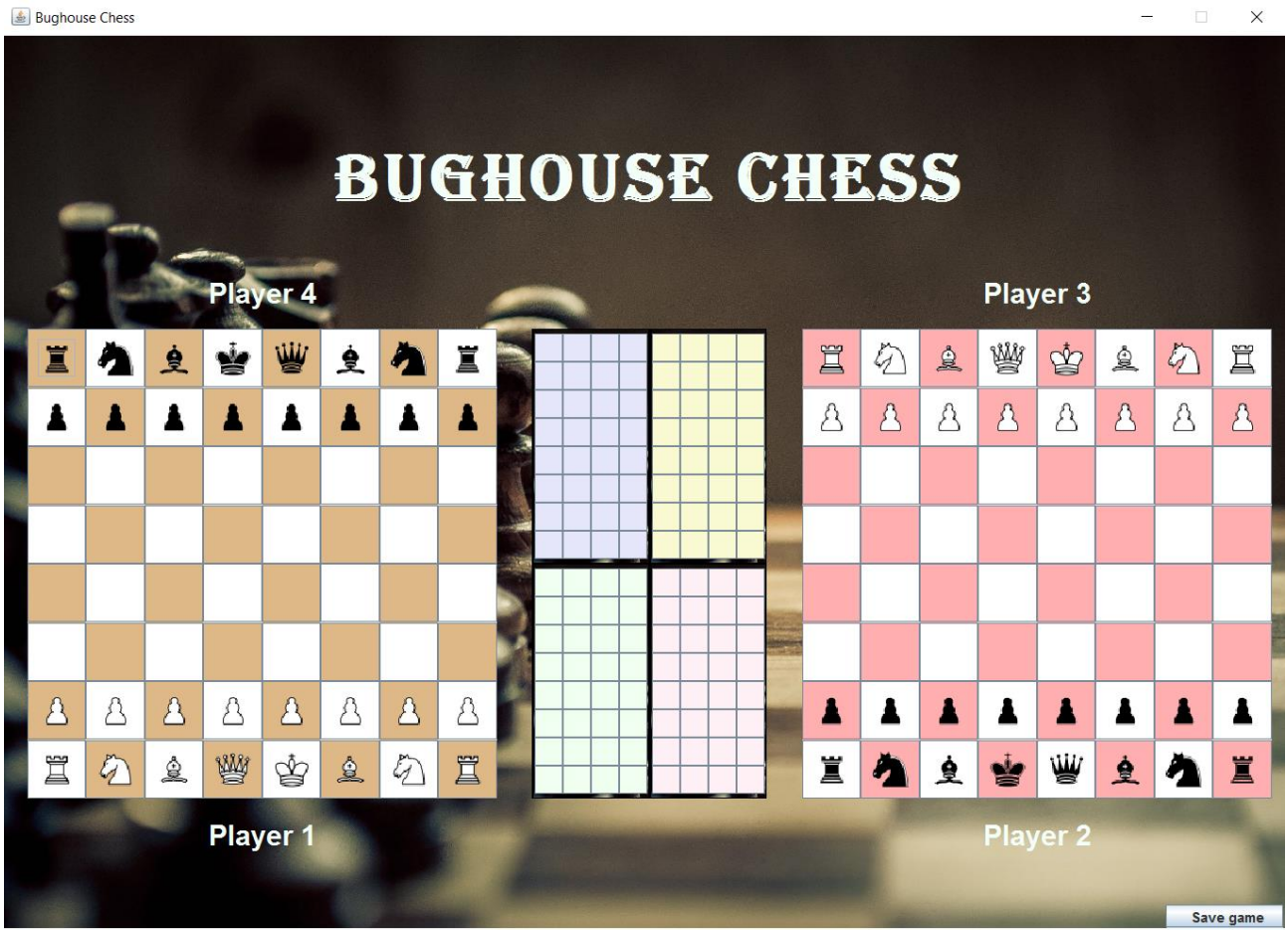


Скріншот меню програми (дизайн все ще в розробці)

Дизайн, запис поточних партій та усе інше реалізовано за допомогою бібліотек:

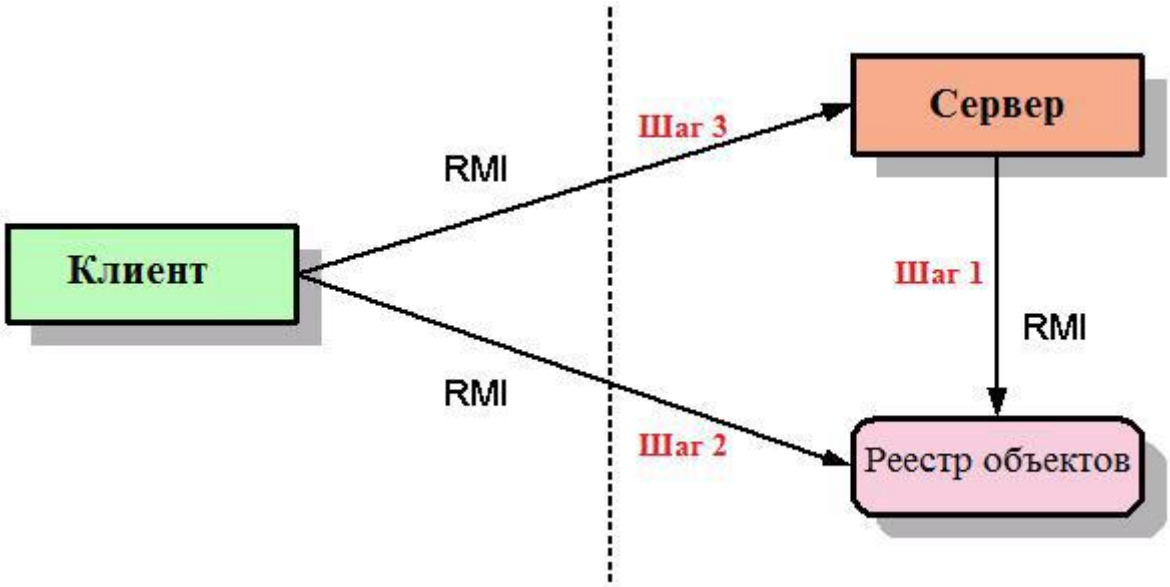
- java.io.IOException
- java.io.ObjectInputStream
- java.io.ObjectOutputStream

- java.net.UnknownHostException
- java.awt
- java.swing

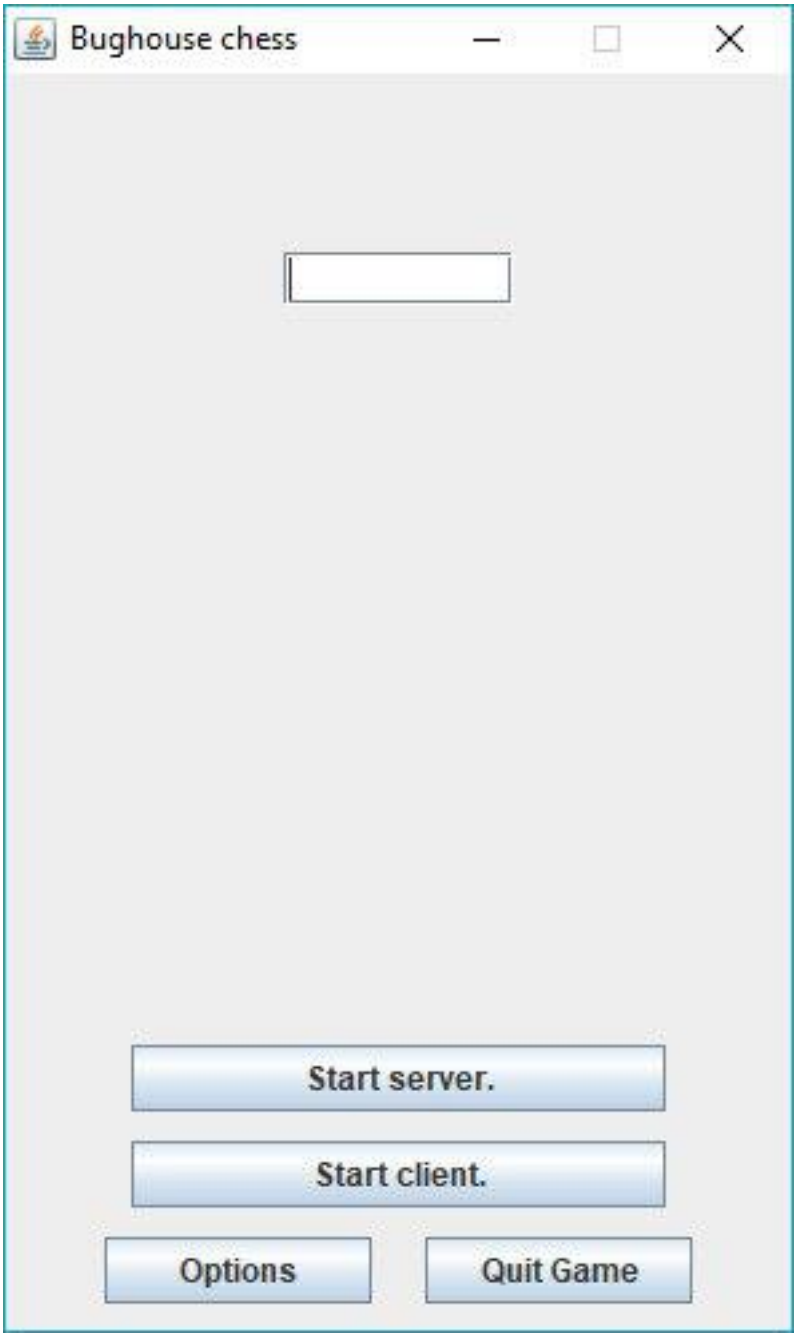


Скріншот грального партії (дизайн все ще в розробці)

Серверна частина сервісу розроблена за допомогою засобів JavaEE. Клієнт-серверна взаємодія здійснюється за домопогою java.net.Socket (відбувається інтеграція з зовнішньою системою). Під час гри через інтернет один гравець створює гру та стає сервером, а інші гравці, які підключаються до нього – клієнтами. Взаємодія гравців реалізована через RMI. RMI (RemoteMethodInvokation) – віддалений виклик методів. Або іншими словами RMI - це механізм, який дозволяє об'єкту в одній Java-машині викликати методи об'єкту в інший Java-машині, навіть якщо вони знаходяться на різних комп'ютерах, в різних країнах, на різних сторонах земної кулі.



Программа сервера містить в собі деякий обєкт, а программа клієнта викликає його методи. Програма містить спеціальний інтерфейс Communicator в якому є інтерфейс-маркер Remote, а також виключення RemoteException. Під час виклика методу можуть траплятися незаплановані збої – тоді буде кидатися це виключення. А класи Client та Server реалізують цей інтерфейс. Працює це наступним чином.



Серверна частина:

У змінній UNIC_BINDING_NAME зберігаємо вигадане нами унікальне ім'я нашого віддаленого об'єкта (об'єкта, який доступний віддалено). Якщо програма шарить кілька об'єктів, у кожного має бути своє унікальне ім'я. Унікальне ім'я нашого об'єкта - «server.reverse».

Об'єкт ReverseImpl доступний віддалено, саме його методи і викликаються.

Потім створили спеціальний об'єкт – реєстр. У ньому ми реєструємо об'єкти, які ми шаримо. Далі ними займається Java-машина. 2099 – це порт (унікальний номер, за яким інша програма може звернутися до нашого реєстру об'єктів).

Тобто щоб звернутися до об'єкта, треба знати унікальний номер реєстру об'єктів (порт), знати унікальне ім'я об'єкта і мати такий же інтерфейс, як і той, який реалізовує віддалений об'єкт.

І створення «заглушки». Заглушка – це спеціальний об'єкт, який приймає інформацію про віддалений виклик, розпаковує її, десеріалізує передані параметри методів і викликає потрібний метод. Потім серіалізуются результат або виключення, якщо воно було, і відсилає все це тому, що викликає.

Потім заглушка реєструється в реєстрі нашого об’єкта під унікальною назвою.

Потім присипляється головний потік. Усі віддалені виклики оброблюються в окремих нитках. Головне, щоб програма у цей час працювала. Так що тут просто присипляємо головну нитку и все.

Клієнтська частина:

Так само створено унікальний об’єкт з таким самим ім’ям як у сервера.

Потім створено «Реєстр віддалених об’єктів». У нього такий самий порт – 2099, він повинен бути таким же як і у реєстру у серверної частини.

Коли отримується об’єкта у реєстру, він являється proxy-об’єктом та приводиться до типу інтерфейсу. Інтерфейс має бути успадкованим від інтерфейсу-маркера Remote.

Нарешті ці методи інтерфейсу викликаються так, ніби об’єкт було створено в цій же програмі.

2. Система атоматичної збірки Ant

ApacheAnt - утиліта для автоматизації процесу складання програмного продукту. Є платформонезалежність аналогом утиліти make, де всі команди записуються в XML-форматі. Ant , на відміну від іншого збирача проектів ApacheMaven, забезпечує імперативну, а не декларативну збірку проекту.

На відміну від make, утиліта Ant повністю незалежна від платформи, потрібна лише наявність на застосовуваній системі встановленого робочого середовища Java –JRE. Відмова від використання команд операційної системи і форма XML забезпечують переносимість сценаріїв.

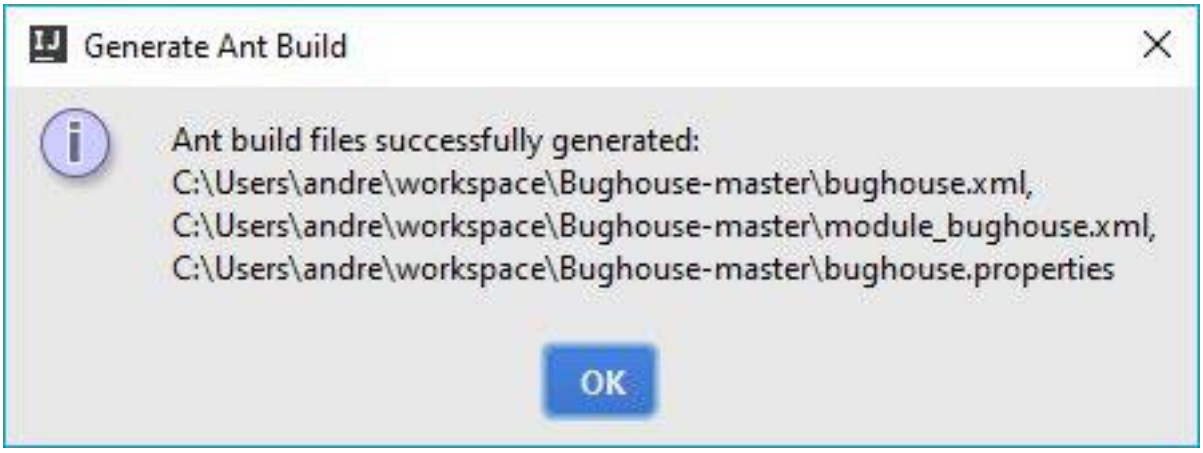
Управління процесом складання відбувається за допомогою XML-сценарію, який також називають Build-файлом. В першу чергу цей файл містить визначення проекту, що складається з окремих цілей (Targets). Цілі можна порівняти з процедурами в мовах програмування і містять виклики команд-завдань (Tasks). Кожне завдання являє собою неподільну, атомарному команду, що виконує деякий елементарне дію.

Між цілями можуть бути визначені залежності - кожна мета виконується тільки після того, як виконані всі цілі, від яких вона залежить (якщо вони вже були виконані раніше, повторного виконання не проводиться).

Типовими прикладами цілей є clean (видалення проміжних файлів), compile (компіляція всіх класів), deploy (розгортання програми на сервері). Конкретний набір цілей і їх взаємозв'язку залежать від специфіки проекту.

Ant дозволяє визначати власні типии завдань шляхом створення Java-класів, що реалізують певні інтерфейси.

Результат виконання для збірки проекту:



3. Сервер безперервної інтеграції.Travis-ci

Це частина процесу розробки, в якій розробляється проект збирається або тестується в різних середовищах виконання автоматично і безперервно. Задумувалася дана методика для найбільш швидкого виявлення помилок та протиріч інтеграції проекту, а отже зниження витрат на наступні простої.

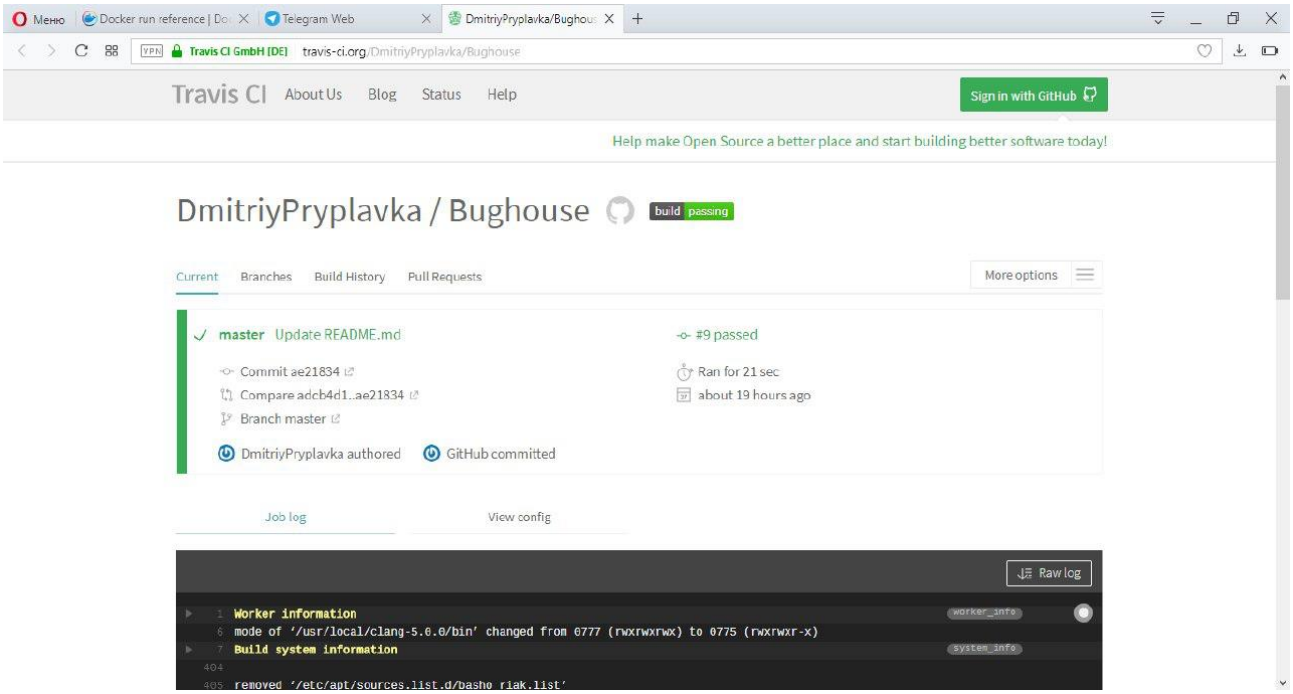
Принцип досить простий: на окремій машині працює якась служба, в обов'язки якої входить отримання вихідного коду проекту, його збірка, тестування, логування, а також можливість надати для аналізу дані виконання перерахованих операцій.

За зручність необхідно платити: виділити окремий сервер і підтримувати його в робочому стані, забезпечити наявність необхідних програмних комплексів, налаштувати середовища виконання, робити резервні копії даних і т.д. Все це вимагає чимало часу і ресурсів. І цілком логічним здається можливість делегувати цю відповідальність на сторонні сервіси. От якраз таким і є travis-ci - «хостинг безперервної інтеграції для opensource співтовариства».

Перелік багів, які шукає утиліта:

- Проблеми продуктивності, пов'язані з розміткою інтерфейсу
- Невикористані ресурси
- Невідповідності розмірів масивів (коли масиви визначені у множинних конфігураціях).

- Проблеми доступності та інтернаціоналізації («магічні» рядки, відсутність атрибуту contentDescription і т.д)
- Проблеми з іконками (невідповідності розмірів, порушення DRY)
- Проблеми зручності використання (Наприклад, не зазначений спосіб введення для текстового поля)
- Помилки в маніфесті



4. Експоненціальна витримка

Для вирішення задачі раптового зникнення з'єднання з базою даних було вирішено задачу експоненціальної витримки (для того, щоб не потрібно було виконувати перезапуск сервера).
Конфігурація представлена нижче:

```
<propertyname = "properties">
      <props>

      <propkey = "hibernate.hbm2ddl.auto">update</prop>
      <propkey = "hibernate.dialect">${jdbc.dialect}</prop>
      <propkey = "hibernate.show_sql">true</prop>
      <propkey = "hibernate.c3p0.minPoolSize">5</prop>
```

```
<propkey = "hibernate.c3p0.maxPoolSize">20</prop>
<propkey = "hibernate.c3p0.idleTestPeriod">1.3</prop>
<propkey = "hibernate.c3p0.timeout">600</prop>
<propkey = "hibernate.c3p0.max_statement">50</prop>
<propkey = "hibernate.c3p0.testConnectionOnCheckout">true</prop>

</props>
</property>
```

Формула витримки складетеся лише з двох параметрів. Мінімальний час у степені кількості спроб. Отриманий графік можна переглянути (нижче на осі x – кількість спроб , на осі y – час затримки до наступної спроби.