**Laporan Kemajuan Tesis**

**Laporan Tesis**

* Revisi berdasarkan hasil seminar:

|  |  |
| --- | --- |
| (Todo) | Cakupan dan batasan penelitian belum diperjelas (masih *source code* Java, bukan Java Web) |
| (Todo) | Penyesuaian judul belum dilakukan. Usulan judul: Pengembangan DSL untuk Verifikasi Formal dan Pengujian pada BDD |
| (Done) | Menambahkan penjelasan mengenai V&V pada bagian latar belakang dan tinjauan pustaka |
| (Done) | Perbaikan rumusan masalah |

|  |  |
| --- | --- |
| **Rumusan Masalah - Awal** | **Hasil perbaikan** |
| Masalah yang dikaji pada penelitian ini adalah verifikasi dan validasi perangkat lunak dengan pendekatan verifikasi formal dan pengujian, yang sulit untuk diselaraskan dengan perubahan spesifikasi dan implementasi perangkat lunak. Diperlukan strategi untuk membuat spesifikasi perangkat lunak yang mencakup kebutuhan untuk verifikasi formal dan pengujian, serta strategi untuk mengekstraksi model formal dari *source code*. Dengan demikian, verifikasi formal dan pengujian akan selalu relevan dengan spesifikasi dan implementasi. | 1. Pengujian yang dilakukan pada proses pengembangan perangkat lunak dengan BDD, tidak cukup untuk menjamin perangkat lunak dari kesalahan (*correctness*). Verifikasi formal dapat melengkapi proses V&V pada BDD. Diperlukan strategi untuk membuat spesifikasi perangkat lunak yang mencakup kebutuhan untuk pengujian dan verifikasi formal. 2. Model perangkat lunak yang telah terverifikasi secara formal belum dapat menjamin *correctness* perangkat lunak, karena sangat tergantung dari implementasinya. Verifikasi formal harus dikenakan pada *source code*, agar hasil verifikasi selalu relevan dan mampu menjadi parameter evaluasi *correctness* perangkat lunak. |

|  |  |
| --- | --- |
| (Done) | Perbaikan tujuan penelitian |
|  | Output penelitian = DSL, dan kakas (DSL *parser*) untuk mentransformasi DSL kedalam interpretasi lain yang mampu digunakan oleh kakas BDD terpilih (JBehave), dan kakas *model checking* |

|  |  |
| --- | --- |
| **Tujuan Penelitian - Awal** | **Hasil perbaikan** |
| Tujuan yang ingin dicapai dari penelitian ini adalah verifikasi dan validasi perangkat lunak dengan dua pendekatan, yaitu verifikasi formal dan pengujian, yang bermanfaat dalam penjaminan *correctness* implementasi perangkat lunak terhadap spesifikasinya. Tujuan spesifik dari penelitian ini adalah menghasilkan teknik yang mampu membangkitkan skrip pengujian dari spesifikasi, mampu memformalkan spesifikasi, dan memodelkan *source code* agar dapat diverifikasi secara formal. | 1. Menghasilkan sebuah *Domain-Specific Language* yang mampu menjadi spesifikasi perangkat lunak, dan mencakup kebutuhan untuk pengujian dan verifikasi formal 2. Menghasilkan kerangka kerja untuk melengkapi proses V&V pada pengembangan perangkat lunak *Behavior-Driven Development*, dengan verifikasi formal yang dikenakan pada level *source code* |

|  |  |
| --- | --- |
| (Done) | Perbaikan metodologi penelitian |

* Tinjauan Pustaka

|  |  |
| --- | --- |
| (Todo) | Belum menambahkan tulisan mengenai hasil eksplorasi kakas *model checking* (*Java Path Finder* dan Bandera), dan eksplorasi terkait anotasi di Java (BSL / *Bandera Specification Language*, JML, *extended* JML) |

* Ekplorasi Kakas
  + Kakas BDD (dibatasi: hanya kakas yang mendukung Java)
    - Cucumber (Cucumber-JVM) 🡪 *Acceptance test*
    - JBehave 🡪 *Acceptance test* & *code generator* (JBehaveCodeGenerator <http://www.mazataz.com/resources/org.custom.jbehave.code.generator_2.0.2.jar> )

*Code generator* (*stub function*) Belum dicoba.

*Stub function* untuk *test*: Baru mencoba yang sederhana

|  |  |
| --- | --- |
| (Todo) | Rencana selanjutnya: melanjutkan eksplorasi pada JBehave. Alasan pemilihan JBehave dijelaskan pada subbab II.7.3 |

* + Kakas untuk anotasi di Java
    - Belum eksplorasi lebih dalam, masih membandingkan:

1. Java Annotation
2. (Todo) JML: JML menggunakan *hoare logic*. Tetapi penelitian (Trentelman dan Huisman, 2002) membuat *extended* JML dengan memasukkan *temporal logic* (penelitiannya dirancang untuk menspesifikasikan properti pada program Java Card)
3. BSL: Bandera Specification Language. melakukan ekstraksi formal model dari *source code*, dan melakukan *model checking* (dengan kakas tambahan yaitu Bogor)
   * Kakas untuk *model checking*
     + Java Path Finder: verifikasi terhadap *executable* Java Bytecode. Digunakan untuk memeriksa *concurrency defect* seperti *deadlocks*, dan *unhandled exception* seperti NullPointerException dan AssertionError

<https://github.com/javapathfinder/jpf-core>

* + - Bandera: tools untuk eksperimen dengan properti *model-checking* pada *source code* Java

Bandera:

1. ekstraksi / konstruksi model dari *source code*
2. menyediakan BSL (*Bandera Specification Language*) dengan *temporal specification language*

Model yang dikonstruksi didefinisikan dalam *guarded command language* yang disebut BIR (*Bandera Intermediate Representation*) . BIR language dapat ditranslasikan ke dalam *input language* pada kakas *model checking* lain (pada paper disebutkan JPF, Spin, SMV, Bogor)

* Analisis dan perancangan

|  |  |
| --- | --- |
| (Done) | Analisis terkait V&V pada existing BDD |

* Rencana Eksperimen

1. Kasus: Simple Withdrawal ATM (<https://github.com/masterthought/jbehave-example> )
2. … Java Web. Kandidat: <https://github.com/mbarbuscio/bow-automation>

On going:

* Ekplorasi kakas BDD, anotasi Java, dan kakas *model checking*
  + Kasus: Simple Withdrawal ATM 🡪 ditest dan diverifikasi
    - *Acceptance test* dengan JBehave
    - Verifikasi dengan Bandera
* Analisis dan perancangan

Referensi

Trentelman, K., dan Huisman, M. (2002): Extending JML Specifications with Temporal Logic, *Algebraic Methodology and Software Technology*, 334–348, Springer, Berlin, Heidelberg, diperoleh melalui situs internet: https://doi.org/10.1007/3-540-45719-4\_23.

**Eksperimen: Simple Withdrawal ATM**

* **Use Case**
* **Class Diagram**
* **State Diagram**
* **JBehave Story**

Narrative: Account Holder withdraws cash

As a Account Holder

I want to withdraw cash from an ATM

In order to get money when the bank is closed

Scenario: Account has sufficient funds

Given the Account balance is <account\_balance>

When the card is valid

When the machine contains <atm\_available>

When the Account Holder requests <request>

Then the ATM should dispense <result>

Then the account balance should be <newBalance>

Then the card should be returned

Examples:

|account\_balance|atm\_available|request|result|newBalance|

|100 |100 |20 |20 |80 |

|100 |100 |10 |10 |90 |

* Kode Pengujian

Configuration File

public class ATMScenario extends JUnitStory {

// Here we specify the configuration, starting from default MostUsefulConfiguration, and changing only what is needed

@Override

public Configuration configuration() {

return new MostUsefulConfiguration()

// where to find the stories

.useStoryLoader(new LoadFromClasspath(this.getClass()))

// CONSOLE and TXT reporting

.useStoryReporterBuilder(new StoryReporterBuilder().withDefaultFormats().withFormats(StoryReporterBuilder.Format.CONSOLE, StoryReporterBuilder.Format.TXT, StoryReporterBuilder.Format.HTML, StoryReporterBuilder.Format.XML));

}

// Here we specify the steps classes

@Override

public List<CandidateSteps> candidateSteps() {

// varargs, can have more that one steps classes

return new InstanceStepsFactory(configuration(), new ATMScenarioSteps()).createCandidateSteps();

}

public class ATMScenarioSteps {

private ATM atm;

private Account account;

private CreditCard creditCard;

private int money;

@Given("the Account balance is <account\_balance>")

public void createAccount(@Named("account\_balance") int balance) {

account = new Account(balance);

}

@When("the card is $valid")

public void createCreditCard() {

creditCard = new CreditCard(account);

}

@When("the machine contains <atm\_available>")

public void createATM(@Named("atm\_available") int money) {

atm = new ATM(money);

}

@When("the Account Holder requests <request>")

public void requestMoney(@Named("request") int amount) {

money = atm.requestMoney(creditCard, amount);

}

@Then("the ATM should dispense <result>")

public void checkMoney(@Named("result") int amount) {

assertThat(money, is(amount));

}

@Then("the account balance should be <newBalance>")

public void checkBalance(@Named("newBalance") int balance) {

assertThat(balance, is(creditCard.getAccount().getBalance()));

}

@Then("the card should be returned")

public void cardShouldBeReturned() {

assertFalse(creditCard.isInUse());

}

}

* Kode Perangkat Lunak

public class Account {

private int balance;

public Account(int balance) {

this.balance = balance;

}

public int getBalance() {

return balance;

}

public int getMoney(int amount) {

if (balance < amount)

return 0;

balance = balance - amount;

return amount;

}

}

public class ATM {

private int money;

public ATM(int money) {

this.money = money;

}

public int requestMoney(CreditCard creditCard, int amount) {

if (!creditCard.isValid() || amount > money) {

return 0;

}

amount = creditCard.getAccount().getMoney(amount);

money = money - amount;

return amount;

}

}

public class CreditCard {

private boolean valid = true;

private boolean inUse = false;

private final Account account;

public CreditCard(Account account) {

this.account = account;

}

public boolean isValid() {

return valid;

}

public Account getAccount() {

return account;

}

public void insertCard(){

inUse = true;

}

public void returnCard(){

inUse = false;

}

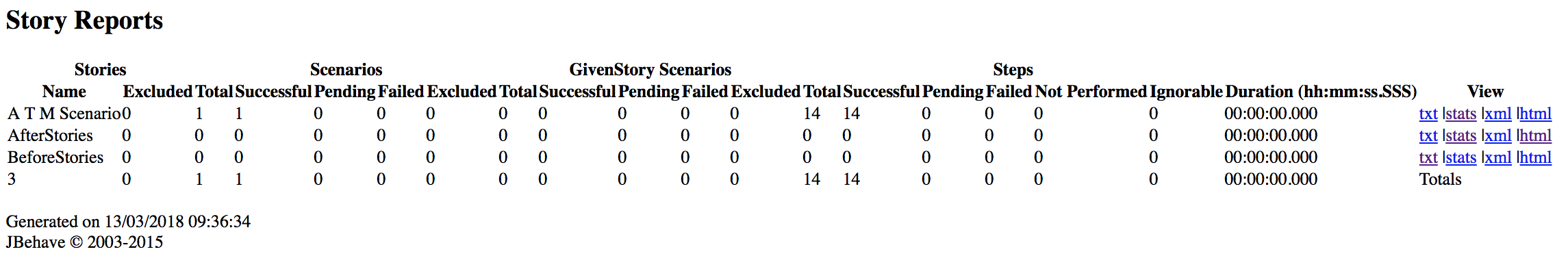
public boolean isInUse() {

return inUse;

}

}

Test Results



* Model Formal (hasil ekstraksi dari Bandera)
* Properti Formal

Eventually ~inUse

Always Balance>=0