FINAL REPORT

FORMAL METHODS IN SOFTWARE ENGINEERING
2018/2019

MASTER IN INFORMATICS AND COMPUTING ENGINEERING

Modeling GitHub in VDM++

Group 9

André Cruz	up201503776@fe.up.pt
Edgar Carneiro	up201503784@fe.up.pt
João Conde	up201503256@fe.up.pt

January 6, 2019

Contents

1 Informal System Description and List of Requirements

This section describes our project's objectives and proposed functionalities, as well as outlining the requirements for the completion of the project.

We developed this project with the VDM++ language [3], and using the Overture tool [4], as is common for formal modelling and validation of information systems [2].

Furthermore, to showcase our system's features we developed a Java client interface, aiming to ease the use of the underlying VDM model.

1.1 Informal System Description

This work aims to model all the information necessary to use and manage the GitHub [1] website, and all of its functionalities.

GitHub is a well known web-based hosting service for version control using Git [5]. It's presented as a development platform to host, manage, and share your projects, and as a driving force for the global open-source community.

Our system allows for *Users* to register on the platform, or to simply navigate the website as a visitor. Registered *Users* may create a *Repository* of their own, be it private or public, and manage the development process using GitHub's functionalities. Additionally, *Users* may follow other *Users*, and may also be members of an Organization.

Repositories have branches associated with them, useful for isolating development work without affecting other branches in the *Repository*. *Commits* are created on a specific *Branch* by a *User* author, and specific commits may be associated with a product *Release*. When appropriate, all work performed on a given *Branch* may be merged onto another *Branch*, re-constructing the *Branch*'s commit history from the sum of the data from the two.

Moreover, *Repositories* have associated *Tags* (*e.g.* VDM, Java), for making it easier to find by the community. There are also a code review and maintenance functionalities, including the creation and assignment of *Issues*, as well as space for discussions.

Finally, our system also allows for general visitors to search the website for *Repositories* with specific *Tags* or a specific name, search for *Users* by username, or list *Repositories* by number of stars (a measure of popularity).

1.2 List of Requirements

These requirements are directly translated onto use cases as shown next (see Section 2).

Id	Priority	Description
R1	Mandatory	As a Visitor, I want to be able to register on the website, to be able to manage my personal projects.
R2	Mandatory	As a Visitor, I want to be able to search the site's repositories by name or tags, to more easily find the projects I'm looking for.
R3	Mandatory	As a Visitor, I want to be able to search the site's users by username, to more easily find the user I'm looking for.
R4	Mandatory	As a User, I want to be able to create a repository.
R5	Mandatory	As a User, I want to be able to follow/unfollow other users, in order to keep myself up to date on others' activities.
R6	Mandatory	As a User, I want to be able to create/delete branches on a given repository, in order to manage its information.
R7	Mandatory	As a User, I want to be able to merge a branch onto another, in order to keep an updated code base.
R8	Mandatory	As a User, I want to be able to star/unstar repositories, in order to save my favourite repositories.
R9	Mandatory	As a User, I want to be able to create releases on a given repository, in order to portray a working version of the product.
R10	Mandatory	As a User, I want to be able to commit to a repository's branch, in order to save my work.
R11	Mandatory	As a User, I want to be able to create issues, in order to identify problems in my repository.
R12	Mandatory	As a User, I want to be able to reopen/close issues, in order to identify which problems were already solved.
R13	Mandatory	As a User, I want to be able to comment on issues, in order to help resolve the related problem.
R14	Mandatory	As a User, I want to be able to edit my account's information, in order to keep it up to date.
R15	Mandatory	As an Owner, I want to be able to set a repository's default branch, in order to manage its state.
R16	Mandatory	As an Owner, I want to be able to assign/remove tags from one of my repositories, in order to better portray its topics.
R17	Mandatory	As an Owner, I want to be able to add collaborators to one of my repositories, in order to manage the team's members.
R18	Mandatory	As an Owner, I want to be able to edit the repository's settings (visibility, description, ...), in order to keep them up to date.

2 Visual UML Model

This section details various UML models [6] to better portray our system's architecture and inner workings.

2.1 Use Case Model

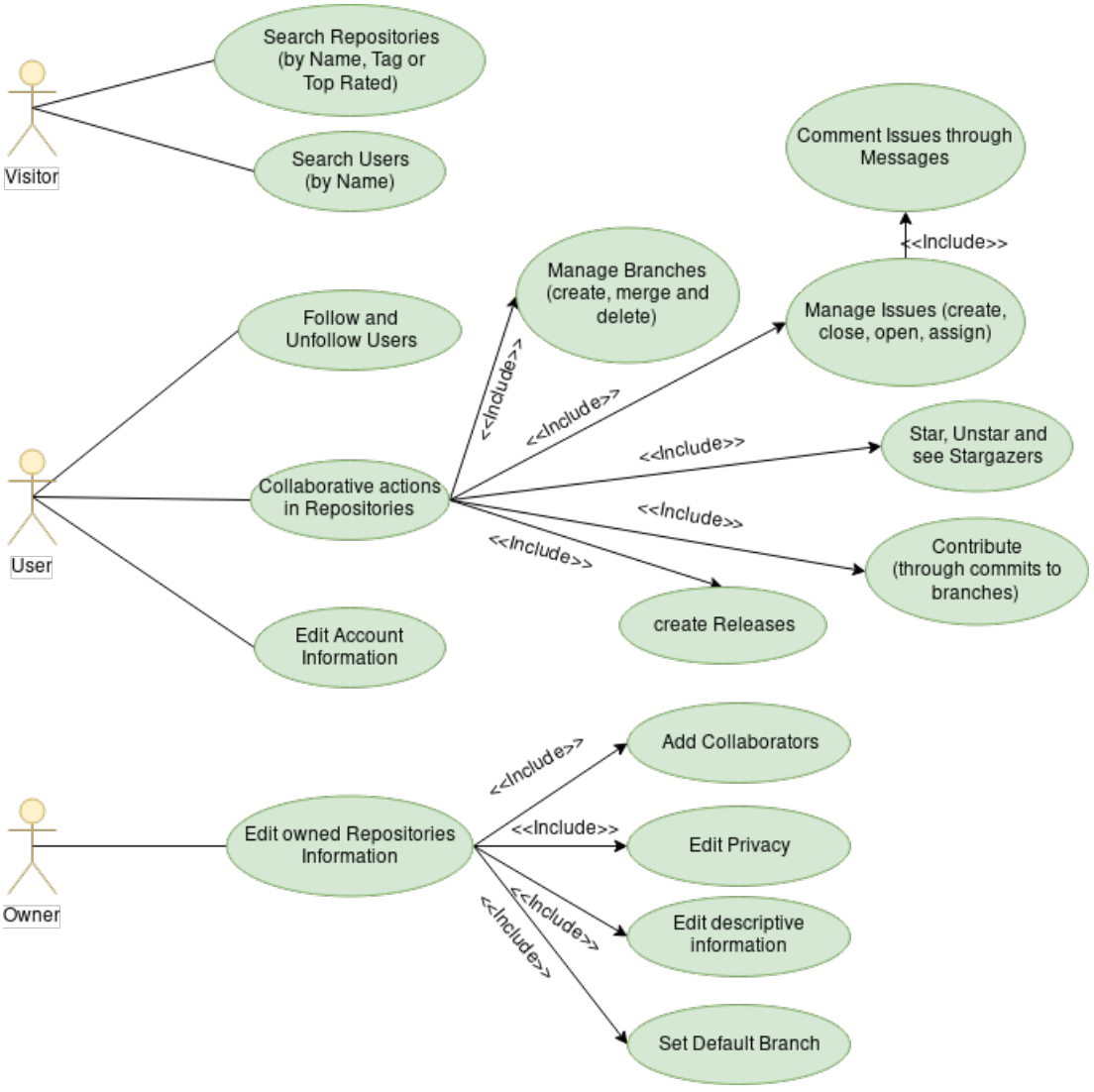

Figure 1: Simplified Use Case Model

Regarding the use case model presented in figure 1 it is important to highlight that an Owner actor can perform all the use cases, while a User actor can perform its use cases as

well as the Visitor's use cases.

It is also important to note that the use case model is slightly simplified making it more easily readable and understandable.

The major use case scenarios are described next:

Scenario	**Create Repository**
Description	Normal Scenario for creating a new repository
Pre-conditions	1. The owner has no repository with the same name. *(initial system state)* 2. The repository name is not empty. *(input)*
Post-conditions	1. The owner received the repository that was created. *(output)* 2. The owner can access the repository through its name. *(final system state)* 3. The repository name, owner and privacy settings are configured as intended. *(final system state)* 4. If the repository owner is a User then the owner is also part of the repository collaborators. *(final system state)* 5. The repository has exactly one branch that is also the default branch and whose name is *master*. *(final system state)*
Steps	1. The User inputs the repository name and privacy settings. 2. The system validates the User input, creates the repository and saves it.
Exceptions	1. The repository name is empty (step 1). 1. The User already has a Repository with the given name (step 1).

Scenario	**Commit**
Description	Normal scenario for making a commit to a repository's branch.
Pre-conditions	1. The Commit name is not empty. *(input)* 2. The User trying to commit to the repository is a collaborator or the repository is public. *(input)* 3. The branch the User is trying to commit to, is one of the repository branches. *(input)* 4. The Commit is not yet part of the branch history. *(initial system state)*
Post-conditions	1. The Commit hash is configured as intended. *(final system state)* 2. The Commit is now part of the branch history. *(final system state)*
Steps	1. The User creates a new Commit. 2. The User commits the new Commit to the related branch. 3. The system validates the new Commit and updates the branch history.
Exceptions	1. The Commit name is empty. (step 1) 2. The repository is not public and the User is not part of the Repository's collaborators. (step 2) 3. The related branch does not belong to the Repository. (step 2) 4. The Commit already belongs to the related branch history. (step 3)

Scenario	**Create Issue**
Description	Normal scenario for creating a Issue in a Repository.
Pre-conditions	1. The User trying to create a new Issue is a collaborator or the owner of the repository. *(input)* 2. There is no Issue in the Repository with the same title. *(initial system state)* 3. The Issue is not in the Repository with other title. *(initial system state)* 4. All the Issue's assignees are collaborators of the related Repository. *(input)*
Post-conditions	1. The owner received the Issue that was created. *(output)* 2. The owner can access the Issue through its title. *(final system state)*
Steps	1. The User creates a new Issue. 2. The User adds the Issue to the related Repository. 3. The system validates the new Issue and updates the Repository Issue's data structure.
Exceptions	1. The user is not a collaborator nor the Repository owner. (step 2) 2. The Repository already has a Issue with an equal title. (step 2) 3. The Repository already has the same Issue with a different title. (step 2) 4. The Issue assignees are not Repoisitory collaborators. (step 2)

Scenario	**Merge Branches**
Description	Normal scenario for merging two branches of a Repository.
Pre-conditions	1. The User trying to create a new Issue is a collaborator of the repository. *(input)* 2. The User can access the target branch through its name, in the Repository. *(initial system state)* 3. The User can access the source branch through its name, in the Repository. *(initial system state)*
Post-conditions	1. If the User chose to delete the source branch on merge, the User can not access it through its name. *(final system state)* 2. If the User chose not to delete the source branch on merge, the User can access it through its name. *(final system state)* 3. The User can access the target branch through its name. *(final system state)* 4. All the commits previously on source branch are now also in the target branch. *(final system state)* 5. All the commits in the target branch are chronologically ordered. *(final system state)*
Steps	1. The User attempts to merge Repository branches. 2. The system validates the destiny and source branch and updates the destiny branches and the Repository branches' data structure.
Exceptions	1. The user is not a collaborator. (step 1) 2. The User can not access the target branch through its name, in the Repository. (step 1) 3. The User can not access the source branch through its name, in the Repository. (step 1)

Scenario	**Search Repositories**
Description	Normal scenario for searching a Repository by its name.
Pre-conditions	(unspecified)
Post-conditions	1. All Repositories found are Repositories belonging to GitHub and are public Repositories. *(output)* 2. All Repositories found have the search query as a substring of their name. *(output)*
Steps	1. The Visitor searches a Repository by its name. 2. The system fetches the satisfying repositories and displays them to the User.
Exceptions	(unspecified)

2.2 Class Model

Initially, after some careful planning, the group decided to implement the following model:

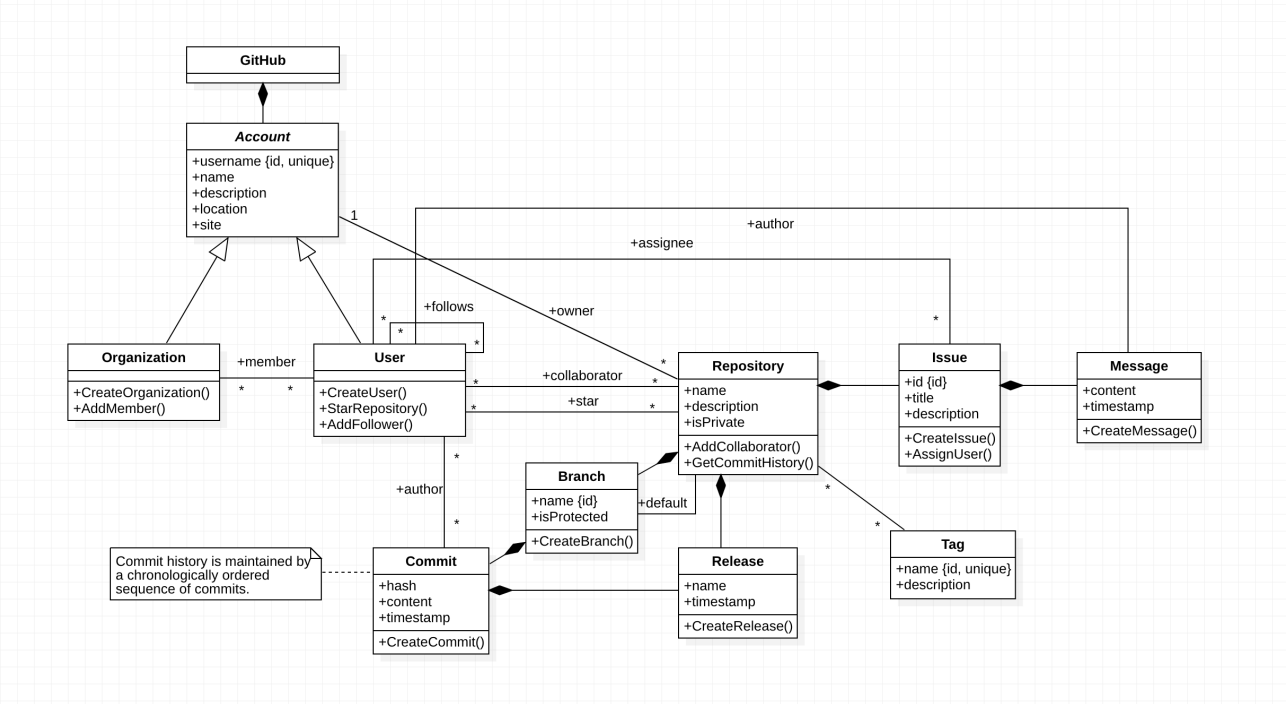

Figure 2: Planned Model

In the end, the result obtained was:

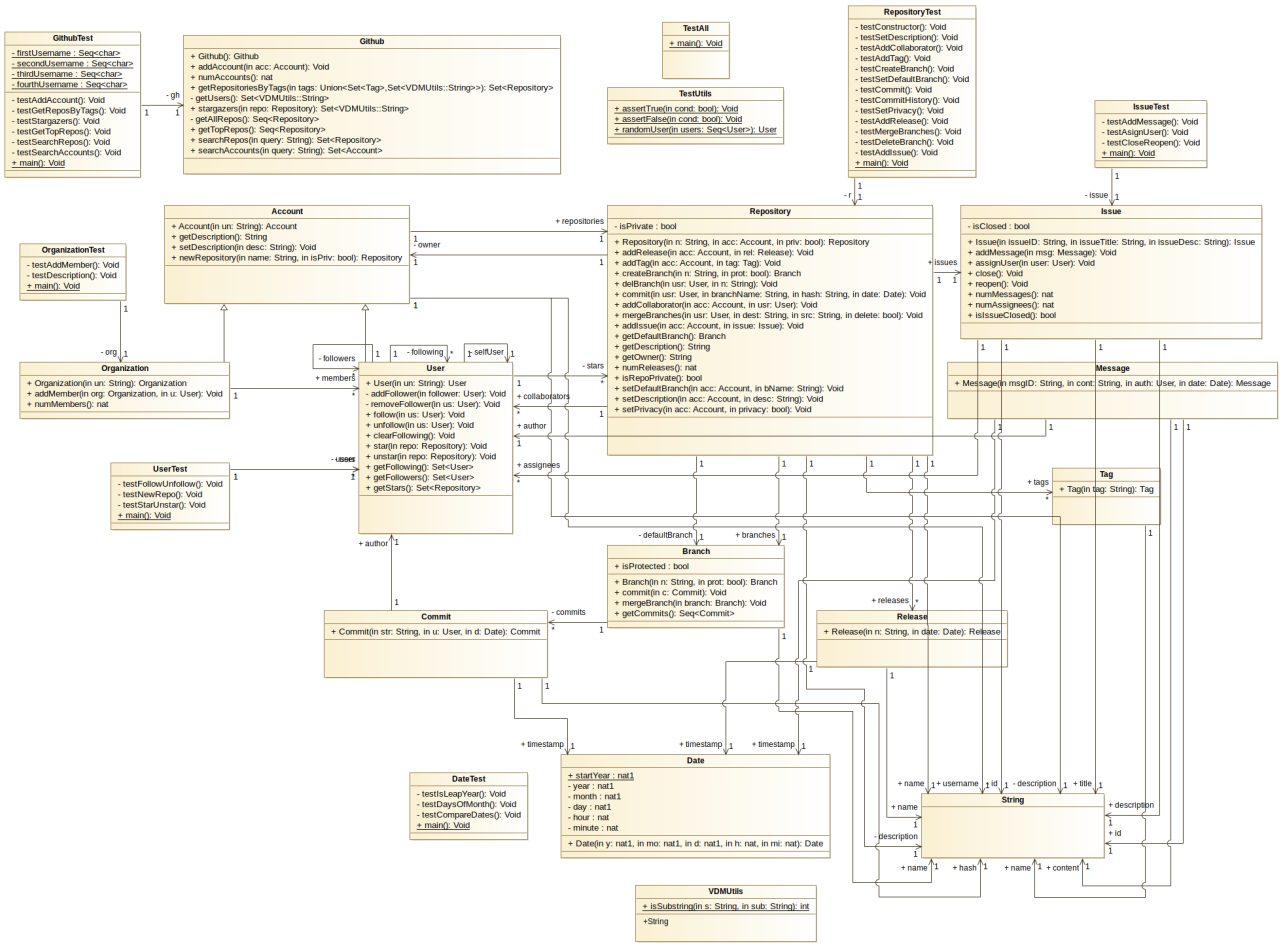

Figure 3: Final Model

Although in the final model the Test Classes and the String type make the model more complex and harder to understand, it is still perceptible the strong resemblance to the planned model.

Class	Description
Account	Defines a GitHub Account and respective operations.
User	Subclass of Account; defines an authenticated User.
Organization	Subclass of Account; defines an Organization.
Github	Code model; defines the operations that allow a User to use Github.
Repository	Defines a GitHub repository as well as all the operations a User can do in it.
Branch	Defines a Branch that will belong to a Repository.
Commit	Defines a way for the User to update a Branch with relevant data.
Release	Defines a working version of the respective Repository code.
Issue	Defines a issue found in a Repository's code.
Message	Defines a Message written by a User in a Issue discussion
Tag	Defines a Tag to be associated with a Repository.
Date	Defines a valid Date and a way of comparing Dates.
VDMUtils	Defines Utility operations, functions and types to be used by all the other classes.
TestAll	Defines the test that calls all the other tests.
GithubTest	Defines the test/ usage scenarios and test cases for GitHub.
RepositoryTest	Defines the test/ usage scenarios and test cases for GitHub Repositories. Also tests Branches and Commits.
IssueTest	Defines the test/ usage scenarios and test cases for GitHub Issues. Also test Messages.
UserTest	Defines the test/ usage scenarios and test cases for GitHub Users. Also tests Accounts.
OrganizationTest	Defines the test/ usage scenarios and test cases for GitHub Organizations. Also tests Accounts.
DateTest	Defines the test/ usage scenarios and test cases for Dates.
TestUtils	Defines utility operations to be used by the other test classes, namely assertTrue and assertFalse.

3 Formal VDM++ Model

3.1 Account

```
class Account

instance variables
 public username: VDMUtils`String;
 public repositories: map VDMUtils`String to Repository := { |-> };

 private description: VDMUtils`String := [];

operations

 /**
  * Constructor for an Account instance.
  */

 public Account: VDMUtils`String ==> Account
 Account(un) == (
  username := un;
 );

 /**
  * Getter for this Account's description.
  */

 public pure getDescription: () ==> VDMUtils`String
 getDescription() == return description;

 /**
  * Setter for this Account's description.
  */

 public setDescription: VDMUtils`String ==> ()
 setDescription(desc) == description := desc
 post description = desc;

 /**
  * Create a new Repository associated with this Account.
  */

 public newRepository: VDMUtils`String * bool ==> Repository
 newRepository(name, isPriv) == (
  let r = new Repository(name, self, isPriv) in (
   repositories(name) := r;
   return r;
```

```
  );
 )
 pre name not in set dom repositories
 post repositories(name).name = name;

end Account
```

Function or operation	Line	Coverage	Calls
Account	14	100.0%	69
getDescription	22	100.0%	3
newRepository	35	100.0%	27
setDescription	28	100.0%	3
Account.vdmpp		100.0%	102

3.2 Branch

```
class Branch

instance variables
 public name: VDMUtils`String;
 public isProtected: bool;

 private commits: seq of Commit := []; -- Chronological order assured by
     invariant

 /**
  * Commits must be chronologically ordered.
  */
 inv forall i1, i2 in set inds commits &
  i1 < i2 => mk_Date`DateComparable(commits(i1).timestamp) < mk_Date`
      DateComparable(commits(i2).timestamp);

operations

 /**
  * Constructor for a Branch instance.
  */

 public Branch: VDMUtils`String * bool ==> Branch
 Branch(n, prot) == (
  name := n;
  isProtected := prot;
```

```
  return self;
 )
 pre n <> []
 post name = n and isProtected = prot;

 /**
  * Add a new Commit to this Branch.
  */

 public commit: Commit ==> ()
 commit(c) == (
  commits := commits ^ [c];
 )
 pre c not in set elems commits
 post c in set elems commits;

 /**
  * Merges another Branch into this Branch by adding non-repeated
  *  Branch Commits to this Branch's Commits and sorting them by Date.
  */

 public mergeBranch: Branch ==> ()
 mergeBranch(branch) == (
  dcl all_commits: seq of Commit := commits ^ [c | c in seq branch.
      getCommits() & (forall co in seq commits & co.hash <> c.hash)];
  dcl sorted_commits: seq of Commit := all_commits;
  for i = len all_commits to 1 by -1 do
   for j = 1 to i-1 do
    if mk_Date`DateComparable(sorted_commits(j).timestamp) >= mk_Date`
       DateComparable(sorted_commits(j+1).timestamp)
     then (dcl temp: Commit := sorted_commits(j);
      sorted_commits(j) := sorted_commits(j+1);
      sorted_commits(j+1) := temp
  );
  commits := sorted_commits;
 )
 post forall c in set elems branch.getCommits() & c in set elems commits;
      -- Commits ordered is assured by invariant

 /**
  * Getter for this Branch's Commits.
  */

 public pure getCommits: () ==> seq of Commit
 getCommits() == return commits;

end Branch
```

Function or operation	Line	Coverage	Calls
Branch	20	100.0%	51
commit	32	100.0%	12
getCommits	61	100.0%	36
mergeBranch	43	100.0%	18
Branch.vdmpp		100.0%	117

3.3 Commit

```
class Commit

instance variables
 public hash: VDMUtils`String;
 public timestamp: Date;

 public author: User;

operations

 /**
  * Constructor for a Commit instance.
  */

 public Commit: VDMUtils`String * User * Date ==> Commit
 Commit(str, u, d) == (
  hash := str;
  author := u;
  timestamp := d;
  return self;
 )
 pre str <> []
 post hash = str;

end Commit
```

Function or operation	Line	Coverage	Calls
Commit	14	100.0%	12
Commit.vdmpp		100.0%	12

3.4 Date

```
class Date
types
 public DateComparable :: date: Date
 ord mk_DateComparable(d1) < mk_DateComparable(d2) ==
  d1.year < d2.year or
  d1.year = d2.year and d1.month < d2.month or
  d1.year = d2.year and d1.month = d2.month and d1.day < d2.day or
  d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.
     hour < d2.hour or
  d1.year = d2.year and d1.month = d2.month and d1.day = d2.day and d1.
     hour = d2.hour and d1.minute < d2.minute;

values
 public static startYear = 2000;

instance variables
 private year: nat1;
 private month: nat1;
 private day: nat1;
 private hour: nat;
 private minute: nat;

 /**
  * Invariants to ensure Date consistency.
  */
 inv year >= startYear;
 inv month >= 1 and month <= 12;
 inv day >= 1 and day <= daysOfMonth(year, month);
 inv hour >= 0 and hour < 24;
 inv minute >= 0 and minute < 60;

operations

 /**
  * Constructor for a Date instance.
  */

 public Date: nat1 * nat1 * nat1 * nat * nat ==> Date
 Date(y, mo, d, h, mi) == (
  year := y;
  month := mo;
  day := d;
  hour := h;
  minute := mi;
  return self;
```

```
 ) -- No need for pre conditions -> assured by instance variables
     invariants
 post year = y and month = mo and day = day and hour = h and minute = mi;

functions

 /**
  * Whether the given year is a leap year.
  */

 public static isLeapYear(y: nat1) res: bool == y mod 4 = 0
 pre y >= startYear;

 /**
  * The number of days in the given month.
  */

 public static daysOfMonth(y, m: nat1) res : nat == (
  cases m :
   1, 3, 5, 7, 8, 10, 12 -> 31,
   4, 6, 9, 11 -> 30,
   2 -> if isLeapYear(y) then 29 else 28
  end
 )
 pre m >= 1 and m <= 12;

end Date
```

Function or operation	Line	Coverage	Calls
Date	35	100.0%	54
daysOfMonth	57	100.0%	6
isLeapYear	51	100.0%	12
Date.vdmpp		100.0%	72

3.5 GitHub

```
class Github

types
 -- Map of String (username) to Account
 public AccountsMap = map VDMUtils`String to Account;

instance variables
```

```
 public accounts: AccountsMap := { |-> };

operations

 /**
  * Constructor for a GitHub instance.
  */

 public Github: () ==> Github
 Github() == (return self)
 post card dom accounts = 0;

 /**
  * Add/register an Account on this GitHub instance.
  */

 public addAccount: Account ==> ()
 addAccount(acc) == (
  accounts(acc.username) := acc;
 )
 pre acc.username not in set dom accounts
 post accounts(acc.username) = acc;

 /**
  * The number of accounts.
  */

 public pure numAccounts: () ==> nat
 numAccounts() == (return card dom accounts);

 /**
  * All repositories whose tags match the given tags.
  */

 public pure getRepositoriesByTags: set of Tag | set of VDMUtils`String
     ==> set of Repository
 getRepositoriesByTags(tags) ==
  return {r | r in set dunion {rng a.repositories | a in set rng accounts
      } & repoMatchesTags(r, tags)}
 pre tags <> {};

 /**
  * Usernames of all registered Users.
  */

 private pure getUsers: () ==> set of VDMUtils`String
 getUsers() == return {un | un in set dom accounts & isofclass(User,
     accounts(un))}
 post forall un in set RESULT & isofclass(User, accounts(un));
```

```
/**
 * Stargares of a given Repository (Users that stared that Repository).
 */

public pure stargazers: Repository ==> set of VDMUtils`String
stargazers(repo) ==
 return {un | un in set getUsers() & repo in set narrow_(accounts(un),
    User).getStars()}
post (forall un in set RESULT & repo in set narrow_(accounts(un), User).
   getStars()) and
 (forall un in set getUsers() \ RESULT & repo not in set narrow_(
    accounts(un), User).getStars());

/**
 * Gets sequence containing all Repositories.
 */

private pure getAllRepos: () ==> seq of Repository
getAllRepos() == (
 dcl reposSet: set of Repository := {}, repos: seq of Repository := [];
 for all acc in set rng accounts do
  reposSet := reposSet union {r | r in set rng acc.repositories \
     reposSet};
 for all r in set reposSet do repos := repos ^ [r];
 return repos;
)
post forall e in set dunion { rng acc.repositories | acc in set rng
   accounts } & e in set elems RESULT and
 card dunion { rng acc.repositories | acc in set rng accounts } = len
    RESULT;

/**
 * Orders Repositories by number of stars using a bubble-sort-like
    algorithm.
 */

public pure getTopRepos: () ==> seq of Repository
getTopRepos() == (
 dcl l: seq of Repository := [r | r in seq getAllRepos() & not r.
    isRepoPrivate()];
 dcl sorted_list: seq of Repository := l;
 for i = len l to 1 by -1 do
  for j = 1 to i-1 do
   if card stargazers(sorted_list(j)) < card stargazers(sorted_list(j+1)
      )
    then (dcl temp: Repository := sorted_list(j);
     sorted_list(j) := sorted_list(j+1);
     sorted_list(j+1) := temp
```

```
  );
  return sorted_list;
 )
 post forall i in set {1, ..., len RESULT - 1} & card stargazers(RESULT(i
     )) >= card stargazers(RESULT(i + 1));

 /**
  * Search Repositories by name.
  */

 public pure searchRepos: VDMUtils`String ==> set of Repository
 searchRepos(query) ==
  return {r | r in set elems getAllRepos() & not r.isRepoPrivate() and
      VDMUtils`isSubstring(r.name, query) >= 0}
 post forall r in set {r | r in set elems getAllRepos() & not r.
     isRepoPrivate()} & if VDMUtils`isSubstring(r.name, query) >= 0 then r
      in set RESULT else r not in set RESULT;

 /**
  * Search Accounts by username.
  */

 public pure searchAccounts: VDMUtils`String ==> set of Account
 searchAccounts(query) ==
  return {accounts(acc) | acc in set dom accounts & VDMUtils`isSubstring(
      acc, query) >= 0}
 post forall acc in set dom accounts & if VDMUtils`isSubstring(acc, query
     ) >= 0 then accounts(acc) in set RESULT else accounts(acc) not in set
      RESULT;

functions

 /**
  * Whether the given Repository matches the given set of Tags.
  */

 public static repoMatchesTags(r: Repository, tags: set of VDMUtils`
     String | set of Tag) res: bool ==
 forall t in set tags & (if isofclass(Tag, t) then t.name else t) in set
     {tInner.name | tInner in set r.tags};

end Github
```

Function or operation	Line	Coverage	Calls
Github	15	100.0%	3

addAccount	22	100.0%	24
getAllRepos	62	100.0%	153
getRepositoriesByTags	38	100.0%	18
getTopRepos	76	100.0%	6
getUsers	46	100.0%	414
numAccounts	32	100.0%	9
repoMatchesTags	112	100.0%	18
searchAccounts	102	100.0%	36
searchRepos	94	100.0%	12
stargazers	53	100.0%	207
Github.vdmpp		100.0%	900

3.6 Issue

```
class Issue

types
 -- Map of message ID to Message
 public Messages = map VDMUtils`String to Message;

instance variables
 private id: VDMUtils`String;
 private isClosed: bool := false;
 public title: VDMUtils`String;
 public description: VDMUtils`String;

 public messages: Messages := { |-> };

 public assignees: set of User := {};

operations

 /**
  * Constructor for Issue instance.
  */

 public Issue: VDMUtils`String * VDMUtils`String * VDMUtils`String ==>
    Issue
 Issue(issueID, issueTitle, issueDesc) == (
  id := issueID;
  title := issueTitle;
  description := issueDesc;
  return self;
```

```
)
post card dom messages = 0;

/**
 * Add a message to this Issue.
 */

public addMessage: Message ==> ()
addMessage(msg) == (
 messages(msg.id) := msg;
)
pre not msg.id in set dom messages
post messages(msg.id) = msg;

/**
 * Assign a User to this Issue.
 */

public assignUser: User ==> ()
assignUser(user) == (
 assignees := assignees union {user};
)
pre not user in set assignees
post user in set assignees;

/**
 * Mark Issue as closed.
 */

public close: () ==> ()
close() == isClosed := true
pre not isClosed
post isClosed;

/**
 * Reopen this Issue.
 */

public reopen: () ==> ()
reopen() == isClosed := false
pre isClosed
post not isClosed;

/**
 * Get the number of messages on this Issue.
 */

public pure numMessages: () ==> nat
numMessages() == return card dom messages;
```

```
 /**
  * Get the number of Assignees of this Issue.
  */

 public pure numAssignees: () ==> nat
 numAssignees() == return card assignees;

 /**
  * Whether this Issue os closed.
  */

 public pure isIssueClosed: () ==> bool
 isIssueClosed() == return isClosed;

end Issue
```

Function or operation	Line	Coverage	Calls
Issue	22	100.0%	9
addMessage	34	100.0%	6
assignUser	44	100.0%	9
close	54	100.0%	3
isIssueClosed	82	100.0%	9
numAssignees	76	100.0%	9
numMessages	70	100.0%	9
reopen	62	100.0%	3
Issue.vdmpp		100.0%	57

3.7 Message

```
class Message

instance variables
 public id: VDMUtils`String;
 public content: VDMUtils`String;
 public timestamp: Date;

 public author: User;

operations
```

```
 /**
  * Constructor for a Message instance.
  */

 public Message:  VDMUtils`String * VDMUtils`String * User * Date ==>
    Message
 Message(msgID, cont, auth, date) == (
  id := msgID;
  content := cont;
  author := auth;
  timestamp := date;
  return self;
 );

end Message
```

Function or operation	Line	Coverage	Calls
Message	15	100.0%	6
Message.vdmpp		100.0%	6

3.8 Organization

```
class Organization is subclass of Account

instance variables
 public members: set of User := {};

operations

 /**
  * Constructor for an Organization instance.
  */

 public Organization: VDMUtils`String ==> Organization
 Organization(un) == (
  Account(un);
 );

 /**
  * Add a member to this Organization.
  */

 public addMember: Organization * User ==> ()
```

```
 addMember(org, u) == (
  members := members union {u}
 )
 pre u.username not in set {us.username | us in set members} and org =
     self
 post u in set members;

 /**
  * Get the number of members of this Organization.
  */

 public pure numMembers: () ==> nat
 numMembers() == return card members;

end Organization
```

Function or operation	Line	Coverage	Calls
Organization	11	100.0%	15
addMember	19	100.0%	6
numMembers	29	100.0%	9
Organization.vdmpp		100.0%	30

3.9 Release

```
class Release

instance variables
 public name: VDMUtils`String := [];
 public timestamp : Date;

operations

 /**
  * Constructor for a Release instance.
  */

 public Release: VDMUtils`String * Date ==> Release
 Release(n, date) == (
  name := n;
  timestamp := date;
  return self;
 )
 pre n <> [];
```

```
end Release
```

Function or operation	Line	Coverage	Calls
Release	12	100.0%	6
Release.vdmpp		100.0%	6

3.10 Repository

```
class Repository

instance variables

 public name: VDMUtils`String;
 private isPrivate: bool;
 private description: VDMUtils`String := [];

 private owner: Account;
 private defaultBranch: Branch;
 public tags: set of Tag := {};
 public collaborators: set of User := {};
 public releases: seq of Release := [];
 public branches: map VDMUtils`String to Branch := { |-> };
 public issues: map VDMUtils`String to Issue := { |-> };

 inv defaultBranch in set rng branches;
 inv branches(defaultBranch.name) = defaultBranch;

 -- All issues' assignees are contributors
 inv forall issue in set rng issues & (forall ass in set issue.assignees
    & ass in set collaborators);

 inv forall i1, i2 in set inds releases &
  i1 < i2 => mk_Date`DateComparable(releases(i1).timestamp) < mk_Date`
     DateComparable(releases(i2).timestamp);

operations

 /**
  * Constructor for a Repository instance.
  */

 public Repository: VDMUtils`String * Account * bool ==> Repository
```

```
Repository(n, acc, priv) == (
 name := n;
 isPrivate := priv;
 owner := acc;
 if isofclass(User, owner) then collaborators := {owner};

 let master = new Branch("master", true) in ( -- Github creates default
     branch master
  defaultBranch := master;
  branches := { "master" |-> master };
 );
 return self;
)
pre n <> []
post name = n and isPrivate = priv and owner = acc and
 (isofclass(Organization, owner) or (isofclass(User, owner) and owner in
     set collaborators)) and
 defaultBranch.name = "master" and card dom branches = 1;

/**
 * Add a Release to this Repository.
 */

public addRelease: Account * Release ==> ()
addRelease(acc, rel) == releases := releases ^ [rel]
pre (if isofclass(User, acc) then narrow_(acc, User) in set
    collaborators else acc = owner) and
 rel.name not in set {r.name | r in seq releases}
post releases(len releases) = rel;

/**
 * Add a Tag to this Repository.
 */

public addTag: Account * Tag ==> ()
addTag(acc, tag) == tags := tags union {tag}
pre acc = owner
post tag in set tags;


public createBranch: VDMUtils`String * bool ==> Branch
createBranch(n, prot) == (
 let b = new Branch(n, prot) in (
  branches(n) := b;
  return b;
 );
)
pre n not in set dom branches
post let b = branches(n) in b.name = n and b.isProtected = prot;
```

```
/**
 * Delete one of this Repository's Branches.
 */

public delBranch: User * VDMUtils`String ==> ()
delBranch(usr, n) == branches := {n} <-: branches
pre usr in set collaborators and
 n <> defaultBranch.name and n in set dom branches --Can not delete
    default branch
post n not in set dom branches;

/**
 * Add a Commit, from a given author, to a given Branch from this
    Repository.
 */

public commit: User * VDMUtils`String * VDMUtils`String * Date ==> ()
commit(usr, branchName, hash, date) == (
 branches(branchName).commit(new Commit(hash, usr, date));
)
pre (usr in set collaborators or not isPrivate) and
 branchName in set dom branches;

/**
 * Add a new collaborator to this Repository.
 */

public addCollaborator: Account * User ==> ()
addCollaborator(acc, usr) == collaborators := collaborators union {usr}
pre acc = owner
post usr in set collaborators;

/**
 * Merges two Branches.
 * Deletes the one being merged if flag delete is set to True.
 */

public mergeBranches: User * VDMUtils`String * VDMUtils`String * bool
   ==> ()
mergeBranches(usr, dest, src, delete) == (
 branches(dest).mergeBranch(branches(src));
 if delete then delBranch(usr, src);
)
pre usr in set collaborators and
 branches(dest) in set rng branches and
 branches(src) in set rng branches
post (if delete then src not in set dom branches else src in set dom
   branches) and
```

```
 dest in set dom branches;

/**
 * Adds an Issue to this Repository.
 */

public addIssue: Account * Issue ==> ()
addIssue(acc, issue) == issues(issue.title) := issue
pre issue.title not in set dom issues and issue not in set rng issues
    and
 ((if isofclass(User, acc) then narrow_(acc, User) in set collaborators
     else acc = owner) or not isPrivate)
post issues(issue.title) = issue;

-- Getters

public pure getDefaultBranch: () ==> Branch
getDefaultBranch() == return defaultBranch;


public pure getDescription: () ==> VDMUtils`String
getDescription() == return description;


public pure getOwner: () ==> VDMUtils`String
getOwner() == return owner.username;


public pure numReleases: () ==> nat
numReleases() == return len releases;


public pure isRepoPrivate: () ==> bool
isRepoPrivate() == return isPrivate;

-- Setters

public setDefaultBranch: Account * VDMUtils`String ==> ()
setDefaultBranch(acc, bName) == defaultBranch := branches(bName)
pre acc = owner and bName in set dom branches
post defaultBranch.name = bName;


public setDescription: Account * VDMUtils`String ==> ()
setDescription(acc, desc) == description := desc
pre acc = owner
post description = desc;
```

```
 public setPrivacy: Account * bool ==> ()
 setPrivacy(acc, privacy) == isPrivate := privacy
 pre acc = owner
 post isPrivate = privacy;

end Repository
```

Function or operation	Line	Coverage	Calls
Repository	31	100.0%	42
addCollaborator	98	100.0%	9
addIssue	121	100.0%	12
addRelease	52	100.0%	6
addTag	61	100.0%	15
commit	88	100.0%	12
createBranch	66	100.0%	9
delBranch	79	100.0%	6
getDefaultBranch	128	100.0%	42
getDescription	131	100.0%	6
getOwner	134	100.0%	1
isRepoPrivate	140	100.0%	162
mergeBranches	107	100.0%	6
numReleases	137	100.0%	9
setDefaultBranch	144	100.0%	6
setDescription	149	100.0%	6
setPrivacy	154	100.0%	6
Repository.vdmpp		100.0%	355

3.11 Tag

```
class Tag

instance variables
 public name: VDMUtils`String;

operations

 /**
  * Constructor for a Tag instance.
  */
```

```
 public Tag: VDMUtils`String ==> Tag
 Tag(tag) == (
  name := tag;
  return self;
 );

end Tag
```

Function or operation	Line	Coverage	Calls
Tag	11	100.0%	12
Tag.vdmpp		100.0%	12

3.12 User

```
class User is subclass of Account

instance variables
 private selfUser: User; -- For invariant purposes
 private followers: set of User := {};
 private following: set of User := {};
 private stars: set of Repository := {};

 inv selfUser not in set followers; -- makes pre self <> user unecessary
 inv selfUser not in set following; -- makes pre self <> user unecessary

operations

 /**
  * Constructor for a User instance.
  */

 public User: VDMUtils`String ==> User
 User(un) == (
  selfUser := self;
  Account(un);
 );

 /**
  * Add a new follower.
  */

 private addFollower: User ==> ()
```

```
addFollower(follower) == (
 followers := followers union {follower};
)
pre follower <> selfUser
post follower in set followers;

/**
 * Remove a follower.
 */

private removeFollower: User ==> ()
removeFollower(us) == (
 followers := followers \ {us};
)
post us not in set followers;

/**
 * Mark this User as following another.
 */

public follow: User ==> ()
follow(us) == (
 following := following union {us};
 us.addFollower(self);
)
pre us <> selfUser
post us in set self.following and
   self in set us.followers;

/**
 * Unfollow another User.
 */

public unfollow: User ==> ()
unfollow(us) == (
 following := following \ {us};
 us.removeFollower(self);
)
pre us in set following and self in set us.followers
post us not in set following and self not in set us.followers;

/**
 * Clear all Users that this User follows.
 */

public clearFollowing: () ==> ()
clearFollowing() == (
 for all us in set following
  do unfollow(us)
```

```
)
post card following = 0;

/**
 * Star a Repository.
 */

public star: Repository ==> ()
star(repo) == stars := stars union {repo}
pre repo not in set stars
post repo in set stars;

/**
 * Unstar a Repository.
 */

public unstar: Repository ==> ()
unstar(repo) == stars:= stars \ {repo}
pre repo in set stars
post repo not in set stars;

-- Getters

public pure getFollowing: () ==> set of User
getFollowing() == return following;


public pure getFollowers: () ==> set of User
getFollowers() == return followers;


public pure getStars: () ==> set of Repository
getStars() == return stars;

end User
```

Function or operation	Line	Coverage	Calls
User	17	100.0%	54
addFollower	26	100.0%	3
clearFollowing	66	100.0%	3
follow	44	100.0%	3
getFollowers	93	100.0%	6
getFollowing	90	100.0%	6
getStars	96	100.0%	423

removeFollower	35	100.0%	3
star	76	100.0%	9
unfollow	55	100.0%	3
unstar	84	100.0%	6
User.vdmpp		100.0%	519

3.13 VDMUtils

```
class VDMUtils

types
 public String = seq of char;

operations

 /**
  * Returns index where the first substring match starts or -1 if it is
      not substring.
  */

 public pure static isSubstring: String * String ==> int
 isSubstring(s, sub) == (
  if len s < len sub
   then return -1;
  for i = 1 to len s - len sub + 1 do
   if s(i, ..., i + len sub - 1) = sub
    then return i;
  return -1;
 )
 post if RESULT >= 0 then s(RESULT, ..., RESULT + len sub - 1) = sub else
      RESULT = -1;

end VDMUtils
```

Function or operation	Line	Coverage	Calls
isSubstring	11	100.0%	174
VDMUtils.vdmpp		100.0%	174

4 Model Validation

4.1 DateTest

```
class DateTest

operations

 private testIsLeapYear: () ==> ()
 testIsLeapYear() == (
  TestUtils`assertTrue(Date`isLeapYear(2020));
  TestUtils`assertFalse(Date`isLeapYear(2018));
 );


 private testDaysOfMonth: () ==> ()
 testDaysOfMonth() == (
  TestUtils`assertTrue(Date`daysOfMonth(2020, 2) = 29);
  TestUtils`assertTrue(Date`daysOfMonth(2018, 2) = 28);
  TestUtils`assertTrue(Date`daysOfMonth(2018, 12) = 31);
  TestUtils`assertTrue(Date`daysOfMonth(2018, 4) = 30);
 );


 private testCompareDates: () ==> ()
 testCompareDates() == (
  TestUtils`assertTrue(
   mk_Date`DateComparable(new Date(2010, 3, 12, 20, 11)) <
   mk_Date`DateComparable(new Date(2012, 4, 21, 23, 56))
  );
  TestUtils`assertTrue(
   mk_Date`DateComparable(new Date(2010, 3, 12, 20, 11)) <
   mk_Date`DateComparable(new Date(2010, 4, 21, 23, 56))
  );
  TestUtils`assertTrue(
   mk_Date`DateComparable(new Date(2010, 3, 12, 20, 11)) <
   mk_Date`DateComparable(new Date(2010, 3, 21, 23, 56))
  );
  TestUtils`assertTrue(
   mk_Date`DateComparable(new Date(2010, 3, 12, 20, 11)) <
   mk_Date`DateComparable(new Date(2010, 3, 12, 23, 56))
  );
  TestUtils`assertTrue(
   mk_Date`DateComparable(new Date(2010, 3, 12, 20, 11)) <
   mk_Date`DateComparable(new Date(2010, 3, 12, 20, 56))
  );
 );
```

```
 public static main: () ==> ()
 main() == (
  let dt = new DateTest() in (
   dt.testIsLeapYear();
   dt.testDaysOfMonth();
   dt.testCompareDates();
  );
 );


end DateTest
```

Function or operation	Line	Coverage	Calls
main	42	100.0%	3
testCompareDates	18	100.0%	3
testDaysOfMonth	10	100.0%	6
testIsLeapYear	4	100.0%	3
DateTest.vdmpp		100.0%	15

4.2 GithubTest

```
class GithubTest

values
 firstUsername = "sample-username";
 secondUsername = "different-username";
 thirdUsername = "other-username";
 fourthUsername = "another-one";

 tagFEUP = new Tag("FEUP");
 tagVDM = new Tag("VDM");

instance variables
 gh : Github := new Github();

operations

 private testAddAccount: () ==> ()
 testAddAccount() ==
 (
```

```
 TestUtils`assertTrue(gh.numAccounts() = 0);
 gh.addAccount(new User(firstUsername));
 TestUtils`assertTrue(gh.numAccounts() = 1);
 gh.addAccount(new Organization(secondUsername));
 TestUtils`assertTrue(gh.numAccounts() = 2);
);


private testGetReposByTags: () ==> ()
testGetReposByTags() ==
(
 let r1 = gh.accounts(firstUsername).newRepository("MFES", false),
   r2 = gh.accounts(secondUsername).newRepository("OVERTURE", false) in
      (
  r1.addTag(gh.accounts(firstUsername), tagFEUP);
  r1.addTag(gh.accounts(firstUsername), tagVDM);
  r2.addTag(gh.accounts(secondUsername), tagVDM);

  TestUtils`assertTrue(r2 not in set gh.getRepositoriesByTags({tagFEUP})
     );
  TestUtils`assertTrue(r2 in set gh.getRepositoriesByTags({tagVDM}));
  TestUtils`assertTrue(r2 in set gh.getRepositoriesByTags({"VDM"}));
 );
);


private testStargazers: () ==> ()
testStargazers() ==
(
 let u3 = new User(thirdUsername), u4 = new User(fourthUsername), r = u3
    .newRepository("VDM++", false) in (
  gh.addAccount(u3); gh.addAccount(u4);
  u3.star(r);
  TestUtils`assertTrue(gh.stargazers(r) = {thirdUsername});
  u4.star(r);
  TestUtils`assertTrue(gh.stargazers(r) = {thirdUsername, fourthUsername
     });
  u3.unstar(r);
  TestUtils`assertTrue(gh.stargazers(r) = {fourthUsername});
 );
);


private testGetTopRepos: () ==> ()
testGetTopRepos() == (
 let tr = gh.getTopRepos() in (
  TestUtils`assertTrue(len tr = 3);
  TestUtils`assertTrue(tr(1).name = "VDM++");
 );
```

```
 let r1 = gh.accounts(secondUsername).newRepository("privateRepo", true)
     ,
 tr = gh.getTopRepos() in (
  TestUtils`assertTrue(len tr = 3); --Private repo does not appear
  TestUtils`assertTrue(tr(1).name = "VDM++");
  TestUtils`assertTrue(r1 not in set elems tr);
 );
);


private testSearchRepos: () ==> ()
testSearchRepos() == (
 let s = gh.searchRepos("VDM++") in (
  TestUtils`assertTrue(card s = 1);
  for all r in set s do
   TestUtils`assertTrue(r.name = "VDM++");
 );

 let r1 = gh.accounts(secondUsername).newRepository("VDM++", false),
 s = gh.searchRepos("VDM++") in (
  TestUtils`assertTrue(card s = 2);
  TestUtils`assertTrue(r1 in set s);
  for all r in set s do
   TestUtils`assertTrue(r.name = "VDM++");
 );

 let r1 = gh.accounts(thirdUsername).newRepository("overture-VDM++2.0",
     false),
 s = gh.searchRepos("VDM++") in (
  TestUtils`assertTrue(card s = 3);
  TestUtils`assertTrue(r1 in set s);
 );

 let r1 = gh.accounts(firstUsername).newRepository("overture-VDM++",
     true),
 s = gh.searchRepos("VDM++") in (
  TestUtils`assertTrue(card s = 3); --Private repo does not appear
  TestUtils`assertTrue(r1 not in set s);
 );
);


private testSearchAccounts: () ==> ()
testSearchAccounts() == (
 TestUtils`assertTrue(card gh.searchAccounts("username") = 3); --first,
     second and third Users
 TestUtils`assertTrue(card gh.searchAccounts(firstUsername) = 1); --
     first
```

```
  TestUtils`assertTrue(card gh.searchAccounts("other") = 2); --third and
      forth Users
 );


 public static main: () ==> ()
 main() ==
 (
  let gt = new GithubTest() in (
   gt.testAddAccount();
   gt.testGetReposByTags();
   gt.testStargazers();
   gt.testGetTopRepos();
   gt.testSearchRepos();
   gt.testSearchAccounts();
  );
 );


end GithubTest
```

Function or operation	Line	Coverage	Calls
main	105	100.0%	3
testAddAccount	16	100.0%	3
testGetReposByTags	26	100.0%	3
testGetTopRepos	55	100.0%	3
testSearchAccounts	98	100.0%	3
testSearchRepos	69	100.0%	9
testStargazers	41	100.0%	9
GithubTest.vdmpp		100.0%	33

4.3 IssueTest

```
class IssueTest

instance variables
 issue: Issue := new Issue("#7400", "test title", "test description");

operations

 private testAddMessage: () ==> ()
 testAddMessage() == (
```

```
  TestUtils`assertTrue(issue.numMessages() = 0);
  issue.addMessage(new Message("msgID", "Content", new User("username"),
      new Date(2018, 12, 30, 22, 28)));
  TestUtils`assertTrue(issue.numMessages() = 1);
  issue.addMessage(new Message("msgID2", "Content2", new User("username2"
      ), new Date(2018, 12, 30, 22, 29)));
  TestUtils`assertTrue(issue.numMessages() = 2);
 );


 private testAsignUser: () ==> ()
 testAsignUser() == (
  TestUtils`assertTrue(issue.numAssignees() = 0);
  issue.assignUser(new User("username"));
  TestUtils`assertTrue(issue.numAssignees() = 1);
  issue.assignUser(new User("username2"));
  TestUtils`assertTrue(issue.numAssignees() = 2);
 );


 private testCloseReopen: () ==> ()
 testCloseReopen() == (
  TestUtils`assertFalse(issue.isIssueClosed());
  issue.close();
  TestUtils`assertTrue(issue.isIssueClosed());
  issue.reopen();
  TestUtils`assertFalse(issue.isIssueClosed());
 );


 public static main: () ==> ()
 main() == (
  let i = new IssueTest() in (
   i.testAddMessage();
   i.testAsignUser();
   i.testCloseReopen();
  );
 );

end IssueTest
```

Function or operation	Line	Coverage	Calls
main	34	100.0%	3
testAddMessage	7	100.0%	3
testAsignUser	16	100.0%	3

testCloseReopen	25	100.0%	3
IssueTest.vdmpp		100.0%	12

4.4 OrganizationTest

```
class OrganizationTest

instance variables
 org : Organization := new Organization("FEUP");
operations

 private testAddMember: () ==> ()
 testAddMember() == (
  TestUtils`assertTrue(org.numMembers() = 0);
  org.addMember(org, new User("sample-username"));
  TestUtils`assertTrue(org.numMembers() = 1);
  org.addMember(org, new User("different-username"));
  TestUtils`assertTrue(org.numMembers() = 2);
 );


 private testDescription: () ==> ()
 testDescription() == (
  org.setDescription("description sample");
  TestUtils`assertTrue(org.getDescription() = "description sample");
 );


 public static main: () ==> ()
 main() ==
 (
  new OrganizationTest().testAddMember();
  new OrganizationTest().testDescription();
 );

end OrganizationTest
```

Function or operation	Line	Coverage	Calls
main	21	100.0%	6
testAddMember	6	100.0%	3
testDescription	15	100.0%	3
OrganizationTest.vdmpp		100.0%	12

4.5 RepositoryTest

```
class RepositoryTest

instance variables
 o : Account := new Organization("feup");
 r : Repository := new Repository("mfes", o, true);
 u : User := new User("contributtor");

operations

 private testConstructor: () ==> ()
 testConstructor() == (
  let org = new Organization("org"), repo = new Repository("test", org,
     true) in (
   TestUtils`assertTrue(repo.name = "test");
   TestUtils`assertTrue(repo.isRepoPrivate());
   TestUtils`assertTrue(card repo.collaborators = 0);
  );

  let repo = new Repository("test2", u, false) in (
   TestUtils`assertTrue(card repo.collaborators = 1);
  );

  TestUtils`assertTrue(r.getDefaultBranch().name = "master");
  TestUtils`assertTrue(card dom r.branches = 1);
  TestUtils`assertTrue(r.branches("master") = r.getDefaultBranch());
 );


 private testSetDescription: () ==> ()
 testSetDescription() == (
  r.setDescription(o, "Projeto de MFES");
  TestUtils`assertTrue(r.getDescription() = "Projeto de MFES");

  r.setDescription(o, "description");
  TestUtils`assertTrue(r.getDescription() = "description");
 );


 private testAddCollaborator: () ==> ()
 testAddCollaborator() == (
  let u2 = new User("two") in (
   r.addCollaborator(o, u);
   TestUtils`assertTrue(r.collaborators = {u});

   r.addCollaborator(o ,u2);
   TestUtils`assertTrue(r.collaborators = {u, u2});
```

```
 );
);


private testAddTag: () ==> ()
testAddTag() == (
 let tag1 = new Tag("AI"), tag2 = new Tag("WebDev") in (
  r.addTag(o, tag1);
  TestUtils`assertTrue(r.tags = {tag1});

  r.addTag(o, tag2);
  TestUtils`assertTrue(r.tags = {tag1, tag2});
 );
);


private testCreateBranch: () ==> ()
testCreateBranch() == (
 let branch = r.createBranch("develop", true), b = r.branches("develop")
     in (
  TestUtils`assertTrue(branch.name = b.name);
  TestUtils`assertTrue(branch.name = "develop");

  TestUtils`assertTrue(branch.isProtected = b.isProtected);
  TestUtils`assertTrue(branch.isProtected);
 );
 TestUtils`assertTrue(card dom r.branches = 2);
);


private testSetDefaultBranch: () ==> ()
testSetDefaultBranch() == (
 TestUtils`assertTrue(r.getDefaultBranch().name = "master");
 r.setDefaultBranch(o, "develop");
 TestUtils`assertTrue(r.getDefaultBranch().name = "develop");
);


private testCommit: () ==> ()
testCommit() == (
 let usr = new User("contributor"), pub = new Repository("public", o,
    false) in (
  r.addCollaborator(o, usr);
  TestUtils`assertTrue(len r.getDefaultBranch().getCommits() = 0);
  r.commit(usr, r.getDefaultBranch().name, "hash2131434", new Date(2018,
      12, 30, 22, 19));
  r.commit(usr, r.getDefaultBranch().name, "hash2123123", new Date(2018,
      12, 30, 22, 20));
```

```
  r.commit(usr, "master", "hash2123199", new Date(2019, 01, 11, 14, 34))
      ;
  TestUtils`assertTrue(len r.getDefaultBranch().getCommits() = 2);

  -- Can also contribute to public repositories
  pub.commit(usr, pub.getDefaultBranch().name, "hash434567", new Date
      (2018, 12, 30, 22, 21));
  TestUtils`assertTrue(len pub.getDefaultBranch().getCommits() = 1);
 );
);


private testCommitHistory: () ==> ()
testCommitHistory() == (
 TestUtils`assertTrue(len r.getDefaultBranch().getCommits() = 2);
 r.setDefaultBranch(o, "master");
 TestUtils`assertTrue(len r.getDefaultBranch().getCommits() = 1);
);


private testSetPrivacy: () ==> ()
testSetPrivacy() == (
 TestUtils`assertTrue(r.isRepoPrivate());
 r.setPrivacy(o, false);
 TestUtils`assertFalse(r.isRepoPrivate());
);


private testAddRelease: () ==> ()
testAddRelease() ==
 (
  TestUtils`assertTrue(r.numReleases() = 0);
  TestUtils`assertTrue(o.username = r.getOwner());
  r.addRelease(o, new Release("v1.1", new Date(2018, 12, 30, 22, 28)));
  TestUtils`assertTrue(r.numReleases() = 1);
  r.addRelease(u, new Release("v1.2", new Date(2018, 12, 30, 22, 29)));
  TestUtils`assertTrue(r.numReleases() = 2);
 );


private testMergeBranches: () ==> ()
testMergeBranches() == (
 TestUtils`assertTrue(card dom r.branches = 2);
 let c1 = r.branches("develop").getCommits(),
 c2 = r.getDefaultBranch().getCommits() in (
  TestUtils`assertTrue(len c1 = 2);
  TestUtils`assertTrue(len c2 = 1);
  r.mergeBranches(u, "master", "develop", false);
  let mc = r.getDefaultBranch().getCommits() in (
```

```
    TestUtils`assertTrue(forall c in set elems c1 & c in set elems mc);
    TestUtils`assertTrue(forall c in set elems c2 & c in set elems mc);
    TestUtils`assertTrue(len c1 + len c2 = len mc);
    TestUtils`assertTrue(len mc = 3);
   );
  );
  TestUtils`assertTrue(card dom r.branches = 2);

  let branch = r.createBranch("feature", true) in (
   TestUtils`assertTrue(card dom r.branches = 3);
   TestUtils`assertTrue(branch in set rng r.branches);
   r.mergeBranches(u, "develop", "feature", true); --Branch is deleted on
       merge
   TestUtils`assertTrue(card dom r.branches = 2);
   TestUtils`assertTrue(branch not in set rng r.branches);
  );
);


private testDeleteBranch: () ==> ()
testDeleteBranch() == (
 let branch = r.createBranch("other-feature", true) in (
  TestUtils`assertTrue(card dom r.branches = 3);
  TestUtils`assertTrue(branch in set rng r.branches);
  r.delBranch(u, "other-feature");
  TestUtils`assertTrue(card dom r.branches = 2);
  TestUtils`assertTrue(branch not in set rng r.branches);
 );
);


private testAddIssue: () ==> ()
testAddIssue() == (
 let issue: Issue = new Issue("#7400", "test title", "test description")
     in (
  issue.assignUser(u);
  r.addIssue(o, issue); --No problem adding so invariant verified
  TestUtils`assertTrue(card dom r.issues = 1);
  TestUtils`assertTrue(r.issues(issue.title) = issue);
 );

 let pub : Repository = new Repository("pub", o, false),
 issue: Issue = new Issue("#7401", "test title", "test description") in
    (
  pub.addIssue(u, issue); -- Can also add issues to public repos
  TestUtils`assertTrue(card dom pub.issues = 1);
  TestUtils`assertTrue(pub.issues(issue.title) = issue);
 );
);
```

```
 public static main: () ==> ()
 main() == (
  let rt = new RepositoryTest() in (
   rt.testConstructor();
   rt.testSetDescription();
   rt.testAddCollaborator();
   rt.testAddTag();
   rt.testAddRelease();
   rt.testCreateBranch();
   rt.testSetDefaultBranch();
   rt.testCommit();
   rt.testCommitHistory();
   rt.testSetPrivacy();
   rt.testMergeBranches();
   rt.testDeleteBranch();
   rt.testAddIssue();
  );
 );

end RepositoryTest
```

Function or operation	Line	Coverage	Calls
main	170	100.0%	3
testAddCollaborator	35	100.0%	3
testAddIssue	153	100.0%	1
testAddRelease	106	100.0%	3
testAddTag	46	100.0%	3
testCommit	76	100.0%	3
testCommitHistory	92	100.0%	3
testConstructor	9	100.0%	3
testCreateBranch	57	100.0%	3
testDeleteBranch	142	100.0%	1
testMergeBranches	116	100.0%	1
testSetDefaultBranch	69	100.0%	3
testSetDescription	26	100.0%	3
testSetPrivacy	99	100.0%	3
RepositoryTest.vdmpp		100.0%	36

4.6 TestAll

```
class TestAll

operations

 public static main: () ==> ()
 main() ==
 (
  GithubTest`main();
  OrganizationTest`main();
  RepositoryTest`main();
  UserTest`main();
  IssueTest`main();
  DateTest`main();
 );

end TestAll
```

Function or operation	Line	Coverage	Calls
main	4	100.0%	3
TestAll.vdmpp		100.0%	3

4.7 UserTest

```
class UserTest

instance variables
 private users : seq of User := [new User("n") | dummy in set {1, ...,
    5}];
 private user : User := new User("username");

operations

 private testFollowUnfollow: () ==> ()
 testFollowUnfollow() == (
  let u = TestUtils`randomUser(users) in (
   user.follow(u);
   TestUtils`assertTrue(u in set user.getFollowing() and user in set u.
      getFollowers());
   user.clearFollowing();
```

```
   TestUtils`assertFalse(u in set user.getFollowing() or user in set u.
       getFollowers());
  )
 );


 private testNewRepo: () ==> ()
 testNewRepo() == (
  let r = user.newRepository("FEUP-MFES", true) in (
   TestUtils`assertTrue(user.repositories(r.name) = r)
  );
 );


 private testStarUnstar: () ==> ()
 testStarUnstar() == (
  let r = (TestUtils`randomUser(users)).newRepository("FEUP-MFES", true)
      in (
   TestUtils`assertTrue(user.getStars() = {});
   user.star(r);
   TestUtils`assertTrue(user.getStars() = {r});
   user.unstar(r);
   TestUtils`assertTrue(user.getStars() = {});
  );
 );


 public static main: () ==> ()
 main() == (
  let ut = new UserTest() in (
   ut.testFollowUnfollow();
   ut.testNewRepo();
   ut.testStarUnstar();
  );
 );

traces
 FollowUnfollow :
  user.clearFollowing();
  (let u = TestUtils`randomUser(users) in user.follow(u)){1, 5};
  (let u = TestUtils`randomUser(users) in user.unfollow(u)){1, 5};

end UserTest
```

Function or operation	Line	Coverage	Calls
main	36	100.0%	2

testFollowUnfollow	8	100.0%	2
testNewRepo	18	100.0%	2
testStarUnstar	25	100.0%	2
UserTest.vdmpp		100.0%	8

4.8 TestUtils

```
class TestUtils

operations

 public static assertTrue: bool ==> ()
  assertTrue(cond) == return
  pre cond;


  public static assertFalse: bool ==> ()
  assertFalse(cond) == return
  pre not cond;


  public static randomUser: seq of User ==> User
  randomUser(users) == (
   return users(MATH`rand(len users) + 1)
  );

end TestUtils
```

Function or operation	Line	Coverage	Calls
assertFalse	8	100.0%	15
assertTrue	4	100.0%	614
randomUser	12	100.0%	12
TestUtils.vdmpp		100.0%	641

5 Model Verification

Overture generates a total of 239 proof obligations from the VDM++ source code. In the following subsections we demonstrate with selected examples regarding domain verification and invariant verification. Additionally, several tests were made to ensure pre-

conditions/post-conditions/invariants were in fact guaranteeing the model's consistency.

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
27	Github‘getUsers()	legal map application

Table 29: Domain verification.

The code under analysis is:

```
/**
 * Usernames of all registered Users.
 */
private pure getUsers: () ==> set of VDMUtils‘String
getUsers() == return {un | un in set dom accounts & isofclass(User,
    accounts(un))}
post forall un in set RESULT & isofclass(User, accounts(un));
```

The relevant map application is “accounts(un)”, in the set comprehension statement. In this case the proof is trivial, because the quantification “un in set dom accounts” assures the map is accessed only inside its domain.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	PO Name	Type
92	User‘addFollower(User)	state invariant holds

Table 30: Invariant verification.

The code under analysis is:

```
/**
 * Add a new follower.
 */
private addFollower: User ==> ()
addFollower(follower) == (
 followers := followers union {follower};
)
pre follower <> selfUser
```

```
post follower in set followers;
```

The relevant invariant under analysis is:

```
inv selfUser not in set followers; -- (selfUser = self)
```

The proof obligation is the following:

```
(forall follower:User & ((follower <> selfUser) => ((selfUser not in set
   followers) => (selfUser not in set (followers union {follower})))))
```

In this case, the invariant is obviously held, as *follower <>selfUser* and *selfUser not in set followers* naturally imply that *selfUser not in set (followers union* $\{follower\}$*)*.

6 Code Generation

After completing the VDM++ model we began generating code. We only tried and tested the generation of Java code.

At first, given a VDM++ project is structured into multiple classes just like a typical Java project the group expected the code generation to be a relatively simple task with no errors in the generated Java code. However, as we soon realized, this was not the case.

The generated Java code had a few issues. Some were as simple as casting an integer variable to a long one. In these cases we do not understand how the overture tool fails to correct them or avoid them at all. In some cases some comparison methods between objects were not created and so some data types lost important methods that allowed ordering them.

Last but certainly not least, the VDM++ code ensured consistency making use of pre and post conditions aswell as invariants. Such was not assured in the generated Java code, as the previous concepts do not exist. Thus, the Java Command-Line Interface serves the sole purpose of allowing an user to explore the model and interact with it, but does not show the entire complexity of the VDM++ project.

7 Conclusions

The model that was developed fulfills all requirements, and features some additional functionalities and improvements that weren't initially planned. The developed project covers all main functionalities of the GitHub website, and ensures consistency on all operations. We also provided a client interface, written in Java, for showcasing the project's functionalities. Moreover, we achieved a test coverage of 100% on all files, which gives us full confidence on the robustness of our project.

Overall, we think the use of VDM++ was a valuable asset in the development of this project. Even though we weren't initially comfortable with programming in VDM++, as the development process progressed we came to see the clear advantages of using such a tool for formal modelling of a system. Furthermore, we think the code base of a similar project in Java would be considerably larger.

This project took approximately 100 hours, evenly split among the three developers.

References

[1] Build software better, together @GitHub. `https://github.com/`.

[2] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated designs for object-oriented systems*. Springer Science & Business Media, 2005.

[3] Peter Gorm Larsen, Kenneth Lausdahl, Nick Battle, John Fitzgerald, Sune Wolff, Shin Sahara, Marcel Verhoef, Peter WV Tran-Jørgensen, Tomohiro Oda, and Paul Chisholm. Vdm-10 language manual. Technical report, Tech. Rep. TR-001, The Overture Initiative, 2013.

[4] Peter Gorm Larsen, Kenneth Lausdahl, Augusto Ribeiro, Sune Wolff, and Nick Battle. Overture vdm-10 tool support: User guide. *Technical Report TR-2010–02, the Overture Initiative*, 2010.

[5] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[6] Jos B Warmer and Anneke G Kleppe. The object constraint language: Precise modeling with uml (addison-wesley object technology series). 1998.