Methodologies for Software Processes

Seminar 1

Grading

- Seminar activity (3--4 practical assignments):--**50%**
-
-
- Final exam: **-- 50%**
- - Final Written Exam (open books)
- **- To pass the exam you have to obtain minim 5 at the final exam and the final grade is minimum 5**
-

Rules

-
- - seminar activity will be done at the group level
- - groups consist of max 2 students
- final exam is individual and is an open book exam (you can have access at the lecture notes and the seminar notes)

Groups

- - Please form the groups of maximum two colleagues and write your group in file Group.xslx from Teams Class Materials

Course Content

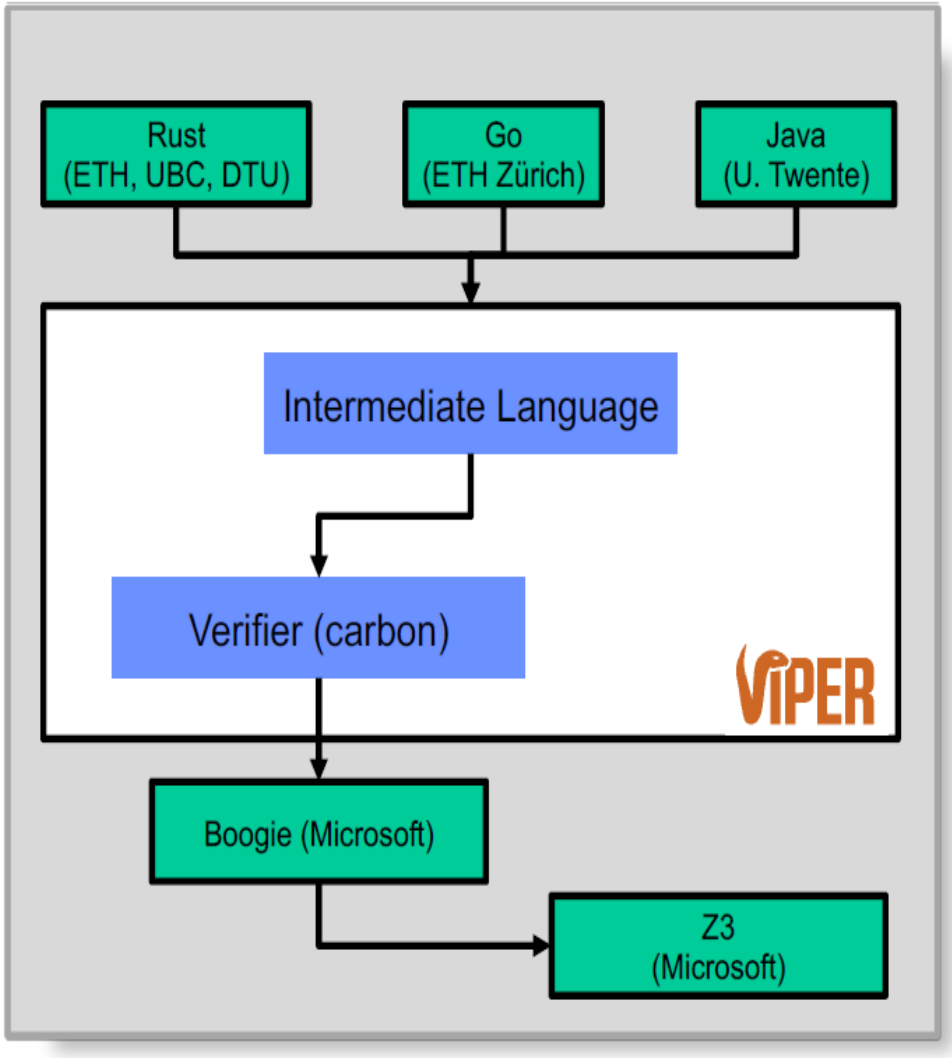
- Please join the Course Teams- code: **kq3h5jq**

- In this course we will discuss about
- **Automated Program Verification**

- using the **VIPER tool**.

The Viper Verification Framework

- Viper language
  - Models verification problems
  - Some statements are not executable
- Two verification backends
  - Carbon (close to what you will build)
  - Silicon
- **For now:** Programming language with a built-in verifier
- **Later:** Automate new methodologies

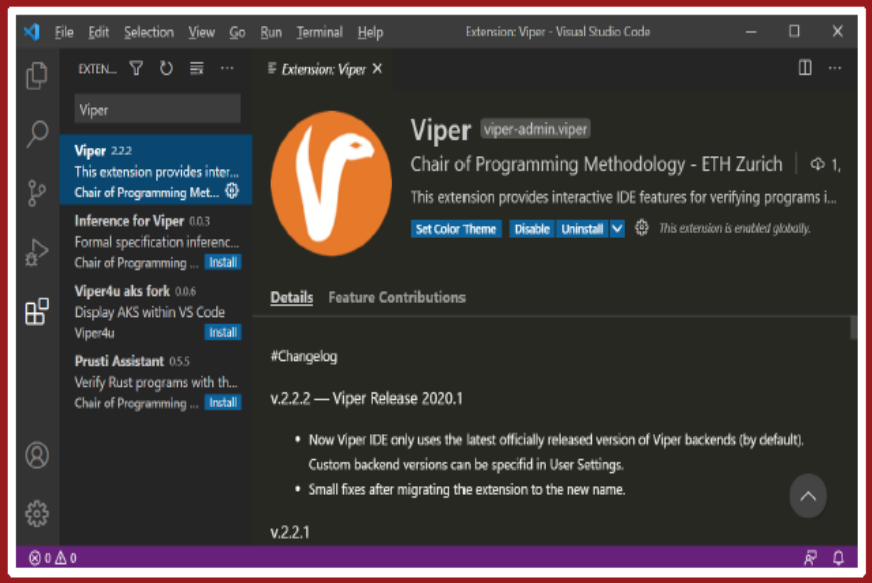


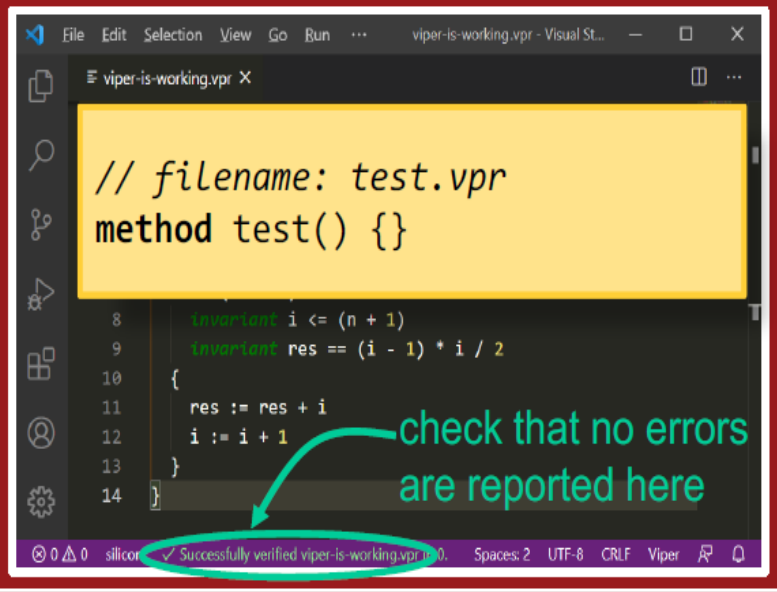
Assignment 1

- Please complete the following tasks until Seminar 2.
- The Assignment 1 must be presented at the Seminar 2 (all group members must be in the class).

Installing Viper

- Install Java 11+ (64-bit)
  - set Java_HOME and PATH
- Install Visual Studio Code (64-bit)
- In Visual Studio Code:
  - Open the extensions browser (⇧+Ctrl+X or ⇧+⌘+X)
  - Search for *Viper*
  - Install the extension and restart
- Create and verify the file test.vpr (right)
- Switch to carbon and verify test.vpr again
  - click on silicon (bottom left) to switch





Viper tutorial

- Please read Lecture 1 and Lecture 2 and the followings:
  - https://viper.ethz.ch/tutorial/#introduction
  - https://viper.ethz.ch/tutorial/#language-overview

Write at least two Viper implementations for the method below that verify.

Try to find one that does *not* compute the maximum.

```
method max(x: Int, y: Int) returns (r: Int)
    ensures r >= x
    ensures r >= y // conjunction of postconditions
{
    // TODO
}
```

Exercise

Consider the method `maxSum` with the following signature:

```
method maxSum(x: Int, y: Int) returns (sum: Int, max: Int)
```

`maxSum` is supposed to store the sum of x and y in variable `sum` and the maximum of x and y in variable `max`, respectively.

a) Define a reasonable contract for `maxSum`.

b) Implement a method that calls `maxSum` on 1723 and 42. Test your contract by adding assertions after the call. Improve your contract if any assertion fails.

c) Implement `maxSum`.

Now, consider a method `reconstructMaxSum` that tries to determine the values of `maxSum`'s input parameters from the output parameters, i.e. it reconstructs x and y from sum and max.

d) Write an abstract method with a postcondition specifying the behaviour of `reconstructMaxSum`.

e) Can you give an implementation of `reconstructMaxSum`? If not, can you implement it after adding a precondition?

f) Write a client to test your implementation of `reconstructMaxSum`.

Forward Reasoning with Viper

- For each of the Viper programs below, replace TODO by the strongest predicate such that the contract verifies; try to find a predicate that is as simple as possible.

- Hint: Use the Viper verifier to check whether the program verifies for your proposed predicates.

```
method a(x: Int, y: Int) returns (z: Int)
    requires 0 <= x && x <= y && y < 100
    ensures TODO
{
    z := y - x
}
```

(b)

```
method b()
{
    var x: Int
    assume 0 <= x && x < 100
    x := 2 * x
    assert TODO
}
```

(c)

```
method c() {
    var x: Int
    var y: Int

    assume x > 0 && x < y

    x := x + 23
    y := y - 3 * x

    assert TODO
}
```

(d)

```
method d() {
   var x: Int
   var y: Bool

   assume x > 0

   x := x + 1
   if (y) {
     var z: Int
     x := x + z
   } else {
     x := 42
   }

   assert TODO
}
```

Backward Reasoning with Viper

- For each of the Viper programs below, replace TODO by the weakest predicate such that the contract verifies; try to find a predicate that is as simple as possible.

- Hint: Use the Viper verifier to check whether the program verifies for your proposed predicates.

(a)

```
method a(x: Int, y: Int) returns (X: Int, Y: Int)
    requires TODO
    ensures X == y && Y == x
{
    X := y - x
    Y := y - X
    X := Y + X
}
```

(b)

```
method b() {

    var x: Int
    var y: Int

    assume TODO

    x := x + y
    y := x * y

    assert x > y
}
```

(c)

```
method c() {
    var x: Int
    var y: Int

    assume TODO

    if (y > 5) {
        y := x - y
    } else {
        x := y - x
    }

    assert x > 7
}
```

(d)

```
method d(x: Int) returns (y: Int)
    requires TODO
    ensures y % 2 == 0
{
    if (x < 17) {
        if (x > 3) {
            y := 1
        } else {
            y := 2
        }
    } else {
        y := 6
    }
}
```