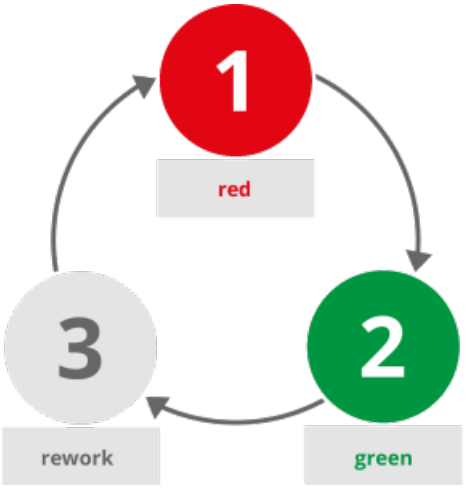
Software development for the pure.box

Test-driven development of Go applications

in the LiteIDE

The LiteIDE is a free development environment for all operating systems - designed for daily working with Go. It support automated tests and debugging using delve or gdb as well as cross-platform compiling. This tutorial explains the test-driven development of Go applications. A small program will be created which adds two numbers together.

Introduction: The test-drive development cycle



1. Red: It all starts with a failed test:

Regardless of whether a new feature was implemented or a bug needs to be fixed: the first step is always to write a test. Since there is as yet no production code, of course it fails.

2. Green: New code meets the test:

Now exactly as much production code is written as needed to successfully carry out the test. The code doesn't have to be particularly beautiful, but should provide as uncomplicated a solution for the single problem.

3. Revise: Ensure readability and maintainability

In the last step the production code is cleaned up, revised and commented in detail. The idea is to keep the resulting code readable and maintainable.

After several iterations in this cycle comprehensive test collections have resulted. In the development of new features all the tests are run. In this way a tight framework for new program sections is prescribed, which in general results in high code quality.

Preparation: Installing the development environment

Have you already installed go? If not, first work through the tutorial "First steps in Go" (Windows) (Linux).

Download the LiteIDE for your operating system from Sourceforge or install it using the package manager. LiteIDE can be extracted into any directory and started from there. Although it already comes with all the necessary tools, it is recommended that additional programs and libraries be installed to expand the functionality:

Guro

```
go get -u golang.org/x/tools/guru
go install golang.org/x/tools/guru
```

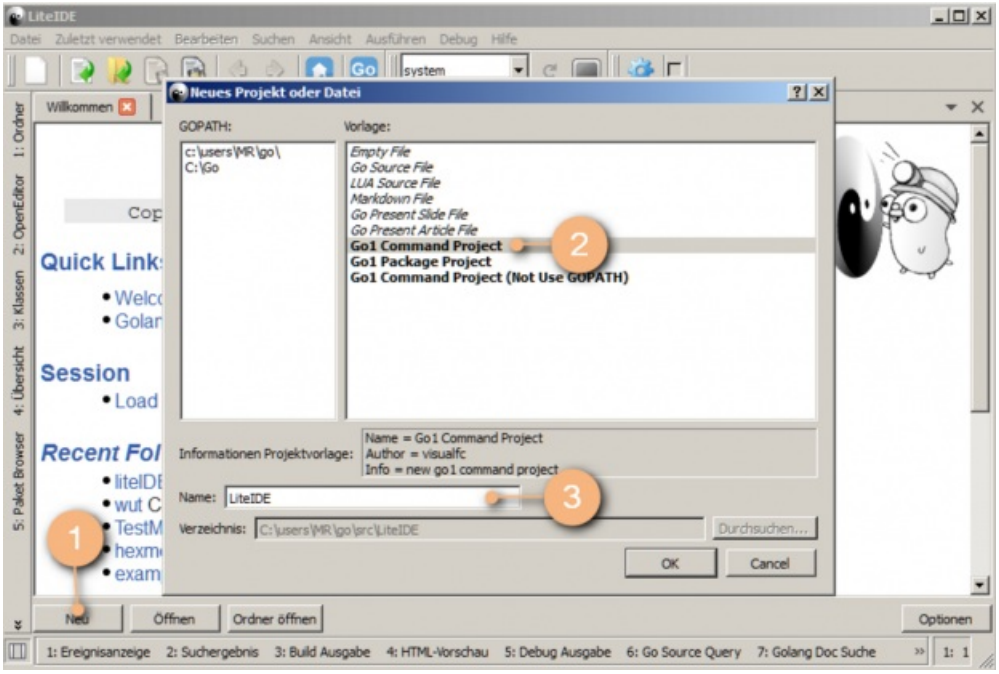
Golint

```
get -u github.com/golang/lint/golint
go install github.com/golang/lint/golint
```

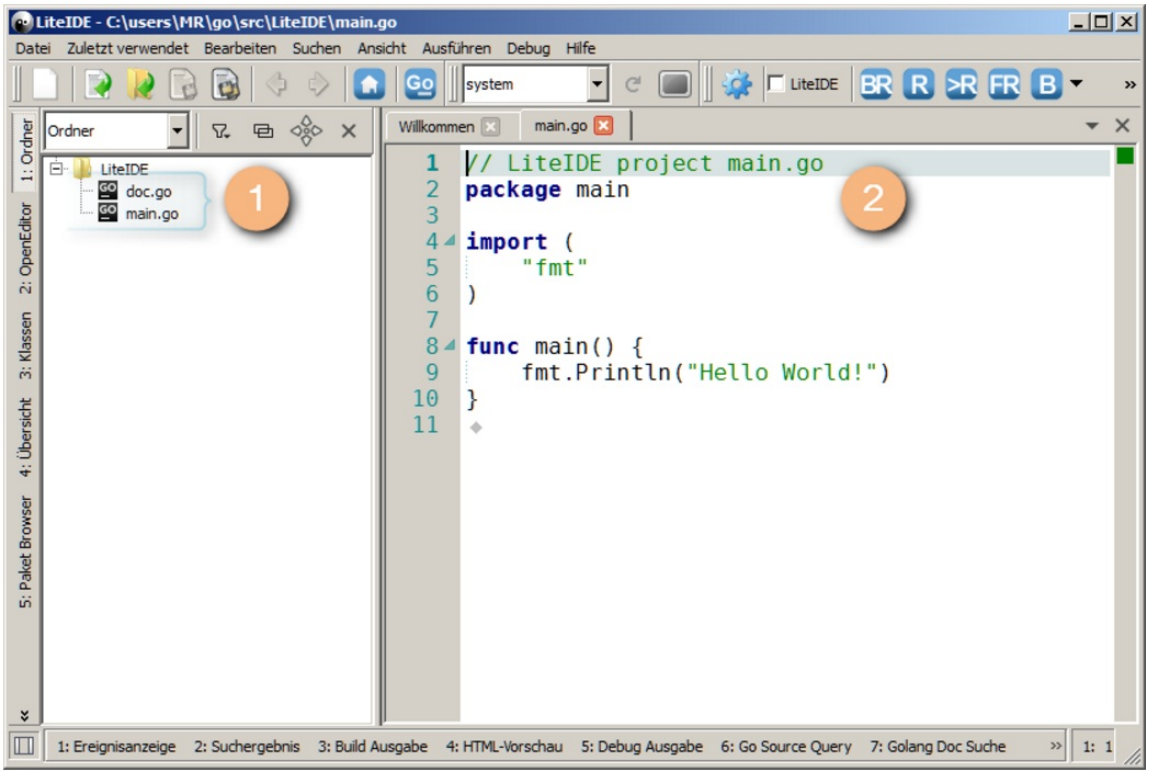
Delve

```
go get -u github.com/derekparker/delve
go install github.com/derekparker/delve
```

Creating a project



1. 1 Click on the "New" button or select "New" from the "File" menu.
2. 2 Select "Go1 Command Project"
3. 3 Give the project a name, for example "litetutorial"



The newly created project already contains two filed: the file **go.doc** 1 , which should contain the project documentation, as well as the file opened in the editor window **main.go** 2 , into owhich the IDE wrote a Hello, World program.

1. Red: Create and run test

In line with the principle "Not a line of code without a failed test," in the first step a test for the function Add(a,b) is created. When the go tool is opened with the command `go test`, it searches in files that end in _test.go for functions that begin with a capitalized test.

Create then the file litetutorial_test.go

```
package main

import "testing"

func TestAdd(t *testing.T) {
  a,b := 2, 3
  want := 5
  result := Add(a, b)

  if result != want {
    t.Errorf("Error in Addition: %d + %d = %d ", a, b, result)
  }
}
```

Line 1:
You want to create an executable file. Therefore the package in which you are working is the main package.

Line 3:
To write a test, use the package `testing` from the standard library.

Line 5:
A function that begins with test is executed as test. As a parameter the function is given a pointer to a `testing.T-Objekt`.

Line 11:
If the function t.Errorf() is called, the test fails. `t.Errorf()` functions similar to the function `fmt.Printf()` or C-function `printf()` and is given a format string as well as various values for the output.

The test is started using the "T" button on the top edge of the screen - or using the key combination Ctrl+T. Since the function `Add(a,b)` does not exist, the test fails.

```
C:/Go/bin/go.exe test -v [C:/Users/Gopher/go/src/litetutorial]
=== RUN TestAdd
--- FAIL: TestAdd (0.00s)
  liteIDE_test.go:12: Error in Addition: 2 + 3 = -1
  FAIL
  exit status 1
  FAIL litetutorial 0.019s
  Fehler: Prozess beendet mit Rückgabewert 1.
```

2. Green: Implement production code and test again

Since the test has failed, the function `Add(a,b)` can now be implemented. To structure the code bases, the function is relocated to the file `calcfunctions.go`.

```
package main
func Add(a, b int) int {
  var result in
  result = a + b
  return result
}
```

After calling the test again using the key combination Ctrl+T the test runs successfully.

3: Cleaning up

The last phase of the test-driven development cycle is easy to underestimate: Clean-up. In this phase the code is reworked so that at the end it is easily understood and maintainable. The program code is thus reduced to its essentials and annotated:

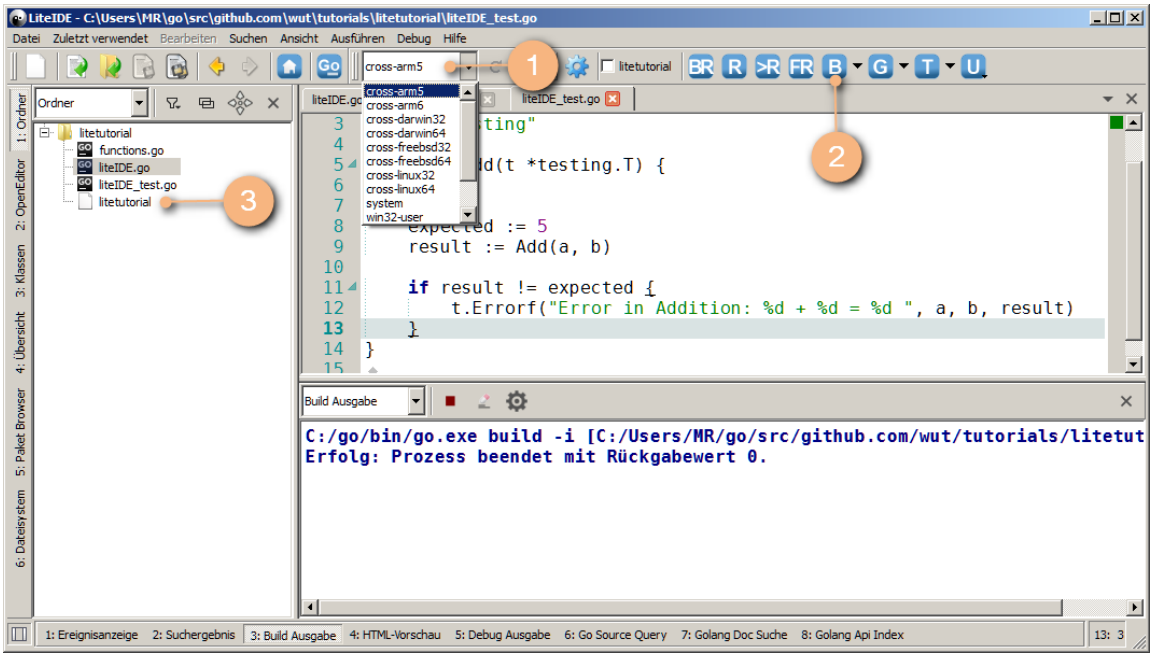
```
//Add returns the sum of to integers a and b
func Add(a, b int) int {
 return a+b
}
```

Another test run is performed without error. The function can now be used in the main program litetutorial.go.

```
package main

import "fmt"

func main() {
 a := 2
 b := 3
 sum := Add(a, b)

 fmt.Println(a, "+", b, "=", sum, "\n")
}
```

A new development cycle can now begin with a test for the subtraction.

Program compiling for the pure.box



1 From the system selection choose "cross-arm5"

2 Clicking on the B-symbol compiles the program

3 The executable file is then included in the project directory.

As shown in the tutorial "First steps in Go", the finished binary can now be loaded into the pure.box via FTP, Samba or SCP and run.

We are available to you in person:

Wiesemann & Theis GmbH
Porschestr. 12
42279 Wuppertal
Phone: +49 202/2680-110 (Mon.-Fri. 8 a.m. to 5 p.m.)
Fax: +49 202/2680-265
info@wut.de

Data Privacy