Software Architectures

Exercise 3

Felix Baumann
Manuel Gottschlich
Alexey Györi 352678
Vincent Wehrwein
Markus Weller

11. Juni 2017

Aufgabe 3.1

a) Formal specification of the requirements

Functionality The software should calculate the inner product of two 3-dimensional vectors.

Interfaces The vectors are specified in the source code, the result is on the command line.

Performance Linear.

Attributes The numbers in the vectors are integers, the result should also be an integer.

b) Flaws of the waterfall model The one main flow of the waterfall model is that is difficult to accomodate change in the model after the process is underway. As software changes can (and in practice always will) occur during the development and implementation process this makes it hard to use in practice. The partitioning of the project into distinct stages is much too inflexible when compared to more flexible patterns like agile programming.

Aufgabe 3.2

a) Perl implementation

```
#!/usr/bin/perl

sub innerProduct
{
        my ($v1_ref, $v2_ref) = @_;

        my @v1 = @$v1_ref;
        my @v2 = @$v2_ref;

        my $result = 0;
        my %values;
        @values{@v1} = @v2;

        foreach my $key (keys %values)
        {
                $result += ($values{$key}*$key);
        }

        return $result;
}

my @vec1 = (1,2,3);
my @vec2 = (3,2,1);

my @result = innerProduct(\@vec1,\@vec2);
print "@result\n";

# prints 10
```

b) Bottom-up vs top-down In the top-down way you would first model the requirements with skeleton functions, interfaces, and data models (ADT). Then, as you progress further and further, you fill the system with function. In the bottom-up way, you start directly with small functions, e.g. for a vector product implement the scalar-multiplication and then extend the functionality of these functioning modules. In very general terms, Object Orientation favors top-down approaches while functional programming favors bottom-up (very very big generalization and absolutely not always true).