**WTI2017 Failure Mechanisms - Dikes Overtopping Kernel**

**Technical Design**

|  |
| --- |
| © Deltares, 2017 |

Edwin Spee

**WTI2017 Failure Mechanisms - Dikes Overtopping Kernel**

**Technical Design**

**WTI2017 Failure Mechanisms - Dikes Overtopping Kernel**

**Technical Design**

Edwin Spee

Date: 17 March 2022

Date: 17 March 2022

Version: 1.4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Client** | | Rijkswaterstaat, WVL | | | | | | | |
| **Title** | | VTV - Dikes overtopping  Technical Design | | | | | | | |
| **Abstract** | | | | | | | | | |
| This document contains the technical design for the dikes overtopping kernel, which forms a part of the WTI 2017 failure mechanism library. The kernel comprises different software components for the calculation of the discharge due to overtopping and the corresponding Z-value.  Dit document bevat het technische ontwerp voor de rekenkern voor overslag bij dijken. Deze rekenkern vormt onderdeel van de bibliotheek van faalmechanismen van WTI 2017 en bestaat uit verscheidene componenten voor het bepalen van het debiet bij overslag en de bijbehorende Z-waarde. | | | | | | | | | |
| **References** | | | |  | | | | | |
| Version | Author | | | Date | Remarks | Review | | Approved by | |
| 1.0 | Edwin Spee | |  | 10-12-2015 | Idem | Erik de Goede |  | A. Baart |  |
| 1.1 | Edwin Spee | |  | 11-12-2015 |  | Erik de Goede |  | A. Baart |  |
| 1.2 | Edwin Spee | |  | 02-06-2016 | extended with thread safety, inverse function and bilingualism | Erik de Goede |  | A. Baart |  |
| 1.3 | Edwin Spee | |  | 01-11-2017 | Extended API for applications in JAVA (Fews) | Erik de Goede |  | M. van Gent |  |
| **Project number** | | |  | | | | | | |
| **Keywords** | | | Dikes overtopping, WTI, Kernel, Technical Design | | | | | | |
| **Number of pages** | | | - | | | | | | |
| **Classification** | | |  | | | | | | |
| **Status** | | | Final | | | | | | |
|  | | | | | | | | | |

wlL5 wlSummary

Contents

wlL2 wlL1 wlL3 wlL4

1 Introduction 3

1.1 Purpose and scope of this document 3

1.2 Other system documents 3

1.3 Assumptions and constraints 4

2 Technical Design 5

2.1 General 5

2.1.1 HydraRing and other Fortran programs 5

2.1.2 RingToets and other C# and .Net programs 5

2.1.3 Dependencies and environment 5

2.2 Description of the required input data for a discharge calculation 6

2.3 Description of the output data for a discharge calculation 7

2.4 Possible error messages 7

2.5 Description of the required input data for the resulting Z-function 7

2.6 Description of the output data for the resulting Z-function 8

2.7 Validation 8

2.8 Version number 8

2.9 GetLanguage and SetLanguage 8

2.10 Inverse function 8

3 Calculation of z2% , the 2% wave run-up. 9

wlListTabs wlListFigs wlListPhotos wlListSymb

# 

# Introduction

## Purpose and scope of this document

This document contains the technical design for the dikes overtopping kernel, which forms a part the WTI 2017 failure mechanism library. The kernel comprises different software components for predicting the discharge due to overtopping and the corresponding Z-function.

Note that the kernel is restricted to wave overtopping. Overflow is not a part of this kernel.

The document will not give any background on the context of the WTI project and on the derivation or motivation of the supported physical model. For this purpose, the reader is referred to the WTI2017 and to its supporting technical reports and their background reports underneath.

This document will describe how the requirements and functional design are implemented in the kernel.

## Other system documents

The full documentation on the kernel comprises the following documents.

|  |  |  |
| --- | --- | --- |
|  | Title | Content |
| 1 | Requirements and functional design | Description of the requirements and functional design. |
| 2 | Technical design | This document |
| 3 | Technical specification | Description of the arguments and usage of different software components, generated from in-line comment with Doxygen |
| 4 | Test plan | Description of the different regression and acceptation rests, including target values. |
| 5 | Test report | Actuated results of the test plan. |

## Assumptions and constraints

1. As a general constraint, the development process needs to comply with the general process description for WTI software, contained in a separate document.
2. As a general constraint, the kernel needs to comply with the relevant general requirements and further design rules for the programming, documentation and testing of WTI software. This set of requirements and rules is contained in a separate document. The set includes the constraints CNS 3 to CNS 5, listed hereafter.
3. As a general WTI software constraint, the failure mechanism library will contain only components for a deterministic analysis to calculate the discharge due to wave overtopping and the corresponding Z-function.
4. As a general WTI software constraint, all appropriate model constants need to be adaptable outside the kernel, in order to allow for varying values during probabilistic analysis.
5. The software interface (API) must allow usage from Fortran (HydraRing) and C# (Ringtoets).
6. To make the overtopping library callable from FEWS, the API must also allow usage from JAVA.

# Technical Design

## General

The dikes overtopping kernel must be usable in Riskeer/Hydra-Ring and other Fortran or C#-programs.

To be able to perform a calculation, regardless of the usage, input parameters are required to define the case to be calculated. An overview of all input data is given in sections 2.2 and 2.5.

### Hydra-Ring and other Fortran programs

In order to be usable in Hydra-Ring, the kernel provides high level functions in the dll[[1]](#footnote-1) for calculating the discharge and the resulting Z-function, where input arguments are Fortran structs[[2]](#footnote-2).

### Riskeer and other C#, .Net and JAVA programs

In order to be usable in Riskeer, the kernel provides high level functions in the dll for calculating the discharge and the resulting Z-function, where input arguments are C# datatypes.

The dll must be wrapped in a C# wrapper class. In the test bench an example wrapper is available that can be used as a blueprint for the actual wrapper in Riskeer. For the moment, such a wrapper will not be part of this kernel.

To be usable in FEWS, the kernel provides high level functions in the dll, where input arguments are datatypes available in JAVA.

### Dependencies and environment

The dikes overtopping kernel is available as a 32-bit Windows native dll, a 64-bit Windows native dll and 64-bit Linux so-file.

There is a (small) dependency on the feedback dll of Hydra-Ring.

The dependencies related to the Intel Fortran compiler are static linked within the dll.

The kernel is thread-safe, in the sense that it can be called by multiple OpenMP threads.

## Description of the required input data for a discharge calculation

To calculate the discharge due to wave overtopping, the user has to provide the hydraulic load, the geometry, the dike height, the model factors, a level for amount of logging and optionally the name of the log file.

Note that the log file is currently only used when the iterations process, described in section 3, ends with a higher residue than expected.

The exact definition of all input structs can be found in the technical documentation (document number 3 in section 1.2). A concise description is given here:

|  |  |  |  |
| --- | --- | --- | --- |
| *parameter* | *Symbol* | *type* | *description* |
| load | H  Hm0  Tm-1,0  ϕ | double  double  double  double | local water level (m + NAP)  significant wave height (m)  spectral wave period (s)  wave direction (degrees in range 0 … 360) |
| geometry |   nPoints  x  y  r | double  integer  double[]  double[]  double[] | orientation of the dike normal (degrees in range 0 … 360)  number of coordinates  xCoords (array of size nPoints, m)  yCoords (array of size nPoints, m + NAP)  roughness (array of size nPoints - 1) |
| dikeHeigth | hcrest | double | dike height (m + NAP) |
| model factors | fn  fb  mz2  fshallow  mq0  mc  Ruser  fred | double  double  double  double  double  double  double  double | model factor for non-breaking waves  model factor for breaking waves  model factor describing the uncertainty of 2% run up height  model factor for shallow waves  model factor computed overtopping discharge  model factor critical overtopping discharge  relaxation factor iteration procedure wave run-up  reduction factor foreshore |
| logging | verbosity  filename | integer  char[255] | one of verboseNone (-1), verboseBasic (0),  verboseDetailed (1), verboseDebugging (2)  filename of the log file |

The struct with model factors is used for both the discharge calculation and the Z-function. The last two entries of the model factors are not actually model factors.

The wave height Hm0 must be >= 0. Small wave heights (Hm0 <= 10-7 *m*) are neglected and lead to zero discharge.

## Description of the output data for a discharge calculation

The calculation of the discharge leads to three types of output:

1. Overtopping output struct
2. Success flag, 0 for success, otherwise failure
3. Error text (only relevant if not successful)

The overtopping output struct consists of two fields: z2 and Qo, respectively 2% wave run-up (m) and the wave overtopping discharge (m3/m per s).

## Possible error messages

The following input is not allowed and will lead to an error message:

1. Local water level is above the crest level (is overflow; not a part of this kernel)
2. Wave height is less than zero
3. Wave period is less than zero
4. Wave direction is not between 0 and 360 degrees
5. The geometry is not valid

See section 2.7 for the possibilities of validation.

Three other error messages may occur:

1. 2% wave run up cannot be calculated, see section 3
2. The model factors must be in a reasonable range and at least >= 0
3. Out of memory

## Description of the required input data for the resulting Z-function

The computation of the Z-value[[3]](#footnote-3) requires three types of input:

1. qc : the critical overtopping discharge rate
2. mc and mq0 , found in the struct with the model factors
3. q0 : the calculated discharge

The struct with the model factors is the same struct as used in the calculation of the discharge, but only the model factors for the computed and critical overtopping are relevant (resp. mq0 and mc).

## Description of the output data for the resulting Z-function

The calculation of the Z-function corresponding to the overtopping discharge, leads to three types of output:

1. Z: the computed Z-value
2. Success flag, 0 for success, otherwise failure
3. Error text (only relevant is not successful)

The Z-function will not be successful only if one of the model factors or the critical overtopping is 0 or negative.

The Z-function is given by:

Where:

|  |  |
| --- | --- |
| mc | Model factor critical overtopping discharge |
| qc | Critical overtopping discharge |
| mq0 | Model factor computed overtopping discharge |
| q0 | Computed overtopping discharge |
|  | Small number (2\*10-306), to avoid |

## Extension of the interface to increase computational efficiency

When running the overtopping kernel in a probabilistic environment, as Hydra-Ring, the kernel is invoked many times, up to 105 – 106 times. Therefore, the computational time must be as short as possible. The original computation uses many allocation statements, which is a relatively expensive operation. To overcome this, we adapted the computation in such a way that only the first invocation allocates memory, and that memory is re-used in all other computations.

To do so, we have to supply a data structure to hold the memory.

The data structure is :

!> tpCoordinatePair: structure with a vector of x and y coordinates

type, public :: tpCoordinatePair

integer :: N !< number of coordinates

real(kind=wp), allocatable :: x(:) !< vector with x-coordinates (m)

real(kind=wp), allocatable :: y(:) !< vector with y-coordinates (m)

end type tpCoordinatePair

!> tpGeometry: structure with geometry data

type, public :: tpGeometry

real(kind=wp) :: psi !< dike normal (degrees)

type(tpCoordinatePair) :: Coordinates !< x and y coordinates cross section

real(kind=wp), allocatable :: roughnessFactors(:) !< roughness factors

type(tpCoordinatePair) :: CoordDiff !< differences in x and y coordinates

real(kind=wp), allocatable :: segmentSlopes(:) !< vector with slopes dike segments

integer, allocatable :: segmentTypes(:) !< vector with segment types

!< (1=slope,2=berm,3=other)

integer :: NbermSegments !< number of berm segments

character :: splitId !< id for B and F splitted type geometries

type(tpGeometries), pointer :: parent !< pointer to all geometries

end type tpGeometry

!> tpGeometries: tree structure with several geometries

type, public :: tpGeometries

type(tpGeometry), pointer :: base

type(tpGeometry), allocatable :: adjWithDikeHeight

type(tpGeometry), allocatable :: geometryMergedBerms

type(tpGeometry), allocatable :: geometrySectionB

type(tpGeometry), allocatable :: geometrySectionF

type(tpGeometry), allocatable :: geometryFlatBerms

type(tpGeometry), allocatable :: geometryNoBerms(:)

type(tpGeometry), allocatable :: geometryRemoveDikeSegments

type(tpCoordinatePair) :: CoordsAdjusted !< vector with x/y-coordinates

!< of the adjusted profile

end type tpGeometries

The extra function that uses this structure is:

call calculateQoHpc ( dikeHeight, OvertoppingInput, resultStruct, load, &

Geometries, error)

The structure must be initialized using:

call setupGeometries(geometryF, Geometries, error)

In this setup routine geometryF is the user input of the geometry and is used to fill Geometries.

And at the end removed using:

call cleanupGeometry(Geometries, .true.)

When using parallel computation using OpenMP, each thread must have each own copy of Geometries.

## Validation

As shown in section 2.4 there are many cases in which the kernel ends up with an error message.

To prevent error messages during the computation, a validation routine is provided in the dll.

The validation only checks the profile and the range of the model factors.

To check the profile, also the dike height must be given.

## Version number

The dll has a function to get the version number of the kernel.

The version number can also be found in the file properties of the dll.

## GetLanguage and SetLanguage

The language for error messages and validation messages can be ‘NL’ or ‘UK’, where the default language is ‘NL’. With the functions GetLanguage and SetLanguage the user can get and set the language to be used in the kernel.

## Inverse function

The inverse function calculates the dike height for a given geometry, load, critical discharge and model factors. In the Overtopping dll, this function is called with its Dutch name: “omkeervariant”.

It first searches at which part of the profile the discharge is near the given discharge. If that section is found it only requires 1 or 2 more evaluations of the discharge, because the logarithmic relation between discharge and dike height can be used.

# Calculation of z2% , the 2% wave run-up.

As given in the functional design, the calculation of the 2% wave run up consists of several steps:

Iterate until 2% wave run-up reaches equilibrium:

a) Estimate 2% wave run-up: provide starting value for z2%

b) Calculate representative slope angle tan .

c) Calculate z2%,smooth, neglecting the effect of berms and roughness (b=1, f=1).

d) Calculate influence factor roughness on slope: f.

e) Calculate z2%,rough, neglecting the effect of berms (assume b = 1).

f) Calculate influence factor berms: b

g) Calculate new influence factor roughness on slope: f.

h) If applicable, adjust the influence factors

i) Calculate 2% wave run-up: z2%

This iteration process can take many iterations and it is not guaranteed that this will converge within a reasonable margin. In some cases, it happens that z2% will flip flop between two values.

To improve convergence, but to change the results as little as possible, the next procedure will be used:

1. Start with z2% = 1.5 Hm0
2. In iteration 2 … 5 use result of previous iteration
3. In iteration 6 … 25 use relaxation only if the user provides a relaxation factor
4. In iteration 26 … 49 use a relaxation factor of at least 0.5
5. Search with 10 small steps at both sides of the value with the lowest residue of the previous steps to find a new minimal residue.

When using relaxation in iterations 6 … 49, the new z2% is given by:

Where:

|  |  |
| --- | --- |
| *R* | Relaxation factor |
|  | z2% at the end of the previous iteration |
|  | z2% at the beginning of the previous iteration |

Note that this approach always leads to a value for z2, but possible with a residue higher than the expected value of 10-3 *m*. In that case a warning is written to the log file, if specified.

Note that within each iteration step, steps a … i may give an error. In that case the computation of z2 fails.

Figure 1 shows a histogram of the number of iterations needed when running the test bench with 32479 evaluations of getting the z2%.

About 95% of the cases need less than 10 iterations. Most of them only need 2 iterations. Approximately 1% finishes shortly after switching on relaxation.

Only one case needs step 5.

All test cases finish without errors in the steps a … i.

|  |
| --- |
| Number of cases  Number of iterations needed |
| Figure 1: number of iterations needed while running the overtopping test bench |

1. A dynamic link library (DLL) is a collection of small programs, any of which can be called when needed by a larger program that is running in the computer [↑](#footnote-ref-1)
2. A struct is a physically grouped list of variables to be placed under one name in a block of memory [↑](#footnote-ref-2)
3. Z-function: limit state function. Z < 0 means failure [↑](#footnote-ref-3)