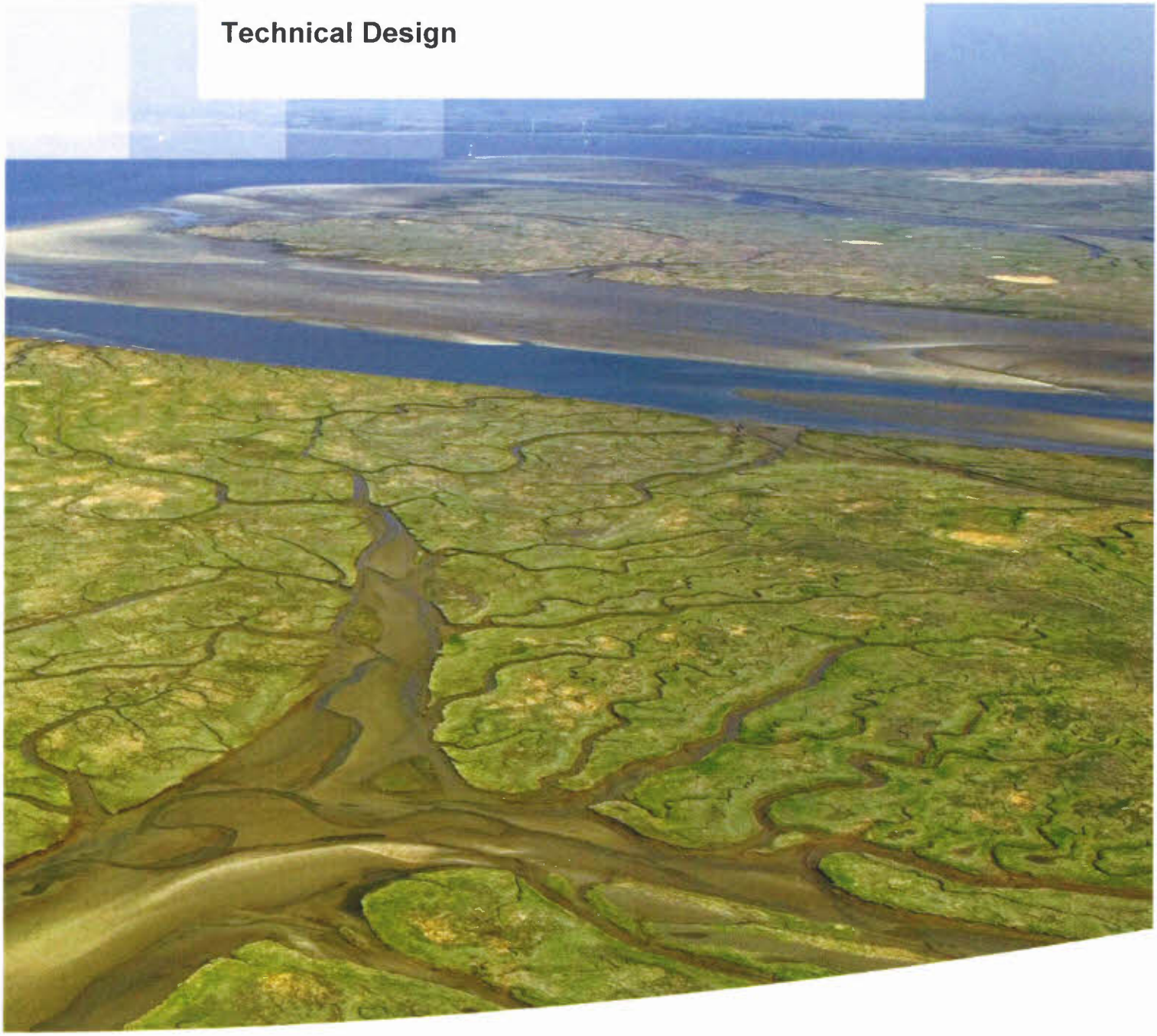
Deltares

Enabling Delta Life

WTI2017 Failure Mechanisms - Piping Kernel

Technical Design



WTI2017 Failure Mechanisms - Piping Kernel

Technical Design

John Bokma

WTI2017 Failure Mechanisms - Piping Kernel

Technical Design

John Bokma

Version: 1.2

Date: 27 July 2015

Client	Waterdienst, Rijkswaterstaat
Title	WTI-2017 Failure mechanisms - Piping kernel Technical Design

Abstract

This document contains the technical design for a so-called piping kernel, which forms eventually a part the WTI 2017 failure mechanism library. The kernel comprises different software components for predicting the occurrence of progressive internal sand erosion in an aquifer under levees and structures. This erosion occurs between the so-called entrance and exit point, after preceding occurrence of uplift of the cover layer (if existing), and provided the occurrence of heave of sand particles at the exit point.

References

Version	Author	Date	Remarks	Review	Approved by
1.1	John Bokma	26-03-2014		Rob Brinkman, Ulrich Förster	Leo Voogt
1.2	John Bokma	24-07-2015		Tom The, Ulrich Förster	Leo Voogt

Project number	1209442.002
Keywords	Piping, Kernel, WTI, Technical Design
Number of pages	27
Classification	
Status	final

Contents

1 Introduction

1.1 Purpose and scope of this document

This document contains the technical design for a so-called piping kernel, which forms eventually a part the WTI 2017 failure mechanism library. The kernel comprises different software components for predicting the occurrence of progressive internal sand erosion in an aquifer under levees and structures. This erosion occurs between the so-called entrance and exit point, after preceding occurrence of uplift of the cover layer (if existing), and provided the occurrence of heave of sand particles at the exit point.

The document will not give any background on the context of the WTI project and on the derivation or motivation of the supported physical models. For this purpose the reader is referred to the VTV2017 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. Note that REQ 5 of the requirements and functional design is not yet to be met so this will not yet be part of this document.

1.2 Other system documents

The full documentation on the piping kernel comprises the following documents.

Title	Content	Authors	Reviewer
Requirements and functional design	Description of the requirements and functional design.	Timo Schweckendiek	John Bokma, Erik Vastenburg
Technical design	This document	John Bokma	Rob Brinkman, Ulrich Förtster, Tom The
Technical specification	Description of the arguments and usage of different software components, generated from in-line comment with Doxygen	John Bokma	Pim Witlox, Ulrich Förster
Test plan	Description of the different regression and acceptation rests, including target values.	John Bokma, Virginie Trompille	Paul Lindhout, Ulrich Förtster
Test report	Actuated results of the test plan.	Virginie Trompille	Pim Witlox, Ulrich Förster

1.3 Assumptions and constraints

CNS 1 As a general constraint, the development process needs to comply with the general process description for WTI software, contained in a separate document [Lit 1].

CNS 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 [Lit 1].

CNS 3 As a WTI software constraint, the failure mechanism library will contain only components for a deterministic analysis to calculate a factor of safety, with a choice between different models for different (sub)mechanisms, that can be called separately. The limit state function (LSF) for probabilistic analysis (using these models) will become a separated part of the mechanism library. In case of different submechanisms, the limit state functions will be supplied only per submechanism. The combination of these submechanisms inside a certain probabilistic procedure is expected to be performed in the external software (notably Hydra-Ring).

CNS 4 As a general WTI software constraint, all model constants need to be adaptable outside the kernel, in order to allow for varying values during probabilistic analysis.

CNS 5 As a WTI software constraint, the failure mechanism library needs to support at least all models that are prescribed for detailed assessment according to the VTV2017.

CNS 6 As a general WTI software constraint, the software interface (API) must allow usage by Ringtoets, Hydra-Ring and MATLAB (test environment).

CNS 7 Besides the prescribed VTV-2017 model, it is assumed that also the VTV2006 piping models for a single aquifer layer need to be supported temporary in the failure mechanism library. The reason is that this will allow a comparison of calculation results during the evaluation period (proeftoets).

CNS 8 The two-layer Sellmeijer model (based on a neural network trained with results from a numerical model) will not be supported in the failure mechanism library. The two-layer Sellmeijer model is compliant to single layer Sellmeijer original[1], and used with PC-Ring by the VNK project. The reason for not supporting it is that it requires a different subsoil model definition, compared to the single layer models. This type of subsoil input will not be supported by the assessment software itself (Ringtoets).

CNS 9 All supported erosion models require a constant thickness of the aquifer layer. Derivation of this value from a 2D soil profile definition will not be part of the kernel.

CNS 10 The location of the "entrance point" is assumed to be user input. The location of the "exit point" is also assumed to be user input. A Calculation Support Module is expected to guide the user in finding the "exit point" and local damping-factor.

CNS 11 Usage of finite element models (FEM) for the determination of the piezometric head in the aquifer, and/or the determination of a heave gradient and/or the direct determination of the critical differential piping head will not be supported by RingToets as an implicit

[1] *When using this two layer VNK model for just one layer with constant thickness, the results become comparable to the original one layer Sellmeijer model.*

part of a detailed analysis in 2017 (level 2a or 2b). External usage is instead assumed to be part of an advanced analysis ("toetsen op maat").

2 Technical Design

2.1 General

As described in the "Requirements and Functional Design", a number of different calculation functions are required. Each of these functions requires its own specific input, performs certain functions and produces its own specific output data. An overview of all the functions is given in section 2.2.

The piping calculator must be usable in RingToets/HydraRing, other C#-programs as well as MatLab (see CNS 6).

2.1.1 RingToets/Hydra-Ring

In order to be usable in RingToets/Hydra-Ring, the calculator provides access to all required data objects via the assembly. This way, all data required to perform a calculation can be set using RingToets. For performing the actual calculations and the retrieval of the requested results, Hydra-Ring will provide delegates itself. A description of the data objects can be found in the technical specification. The mechanism of the delegates is documented in paragraph 2.3 of the RTO Design (Lit 2).

2.1.2 Other C# programs (including MatLab)

Other C# programs can use the same approach as RingToets/Hydra-Ring but can also use the calculation functions defined in the main body of the assembly. A description of the calculation functions can be found in section 2.2.

2.2 Description of the required functions

As described in the "Requirements and Functional Design", a number of different calculation functions are required. As each of these functions requires its own specific input, some input parameters are used in more than one function. Even more, an output parameter of one function can be used as input for another function (e.g. the exit point calculated by the ExitPointDeterminator is used by most other functions as input). For this reason, the specification of all parameters used in the functions is given in annex A.

However, two of parameters (SurfaceLine and SoilProfile) are so called complex types as they are data objects instead of so called simple types (Boolean, integer, string, double, enumeration). Therefore the definition of these types is given in section 2.2.1.

Note that there are some restrictions and requirements:

- all heights and levels must be towards the same reference level (e.g. NAP)
- all coordinates must share the same coordinate system
- Points on the surface line must be defined from left to right with increasing L-coordinates.

For each function a class is designed to house the in- and output parameters as well as the functions (methods) that are to be used to perform the actual actions (calculation and validation).

The general setup per class is that all parameters are defined as properties. All classes hold two functions, one for validation and one for calculation.

In Annex B, examples on using the classes are provided.

2.2.1 Definition of the complex types

The first complex type is the SurfaceLine. The surface line needed by the kernel starts at the toe of the dike at polder side and should run to at least the end of the expected uplift area at polder side.

The SurfaceLine is a data object of the PipingSurfaceLine class. This class has the next input (properties) that have to be provided:

- Name (type = string): the name of the surface line.
- Points (type = List<PipingPoint>): a list of points of type PipingPoint defining the surface line. For the definition of the PipingPoint, see below.

The class PipingPoint has the next properties:

- X (type = double): the x-coordinate of the point in meters.
- Y (type = double): the y-coordinate of the point in meters.
- Z (type = double): the z-coordinate of the point in meters.
- Type (type = PipingCharacteristicPointType): the type of the point (if any). PipingCharacteristicPointType is an enumeration with as possible options None (= default value), ShoulderBaseInside, DikeToeAtPolder, DitchDikeSide, BottomDitchDikeSide, BottomDitchPolderSide, DitchPolderSide.

Note that all points in the surface line must be defined from left to right (so with increasing X-values). Also note that when used, the typed points must be from left to right in the given order of the enumeration (so DikeToeAtPolder must be to the right of the ShoulderBaseInside when both types are used).

Typing points is in fact optional. The kernel handles the surface line as follows:

- DikeToeAtPolder and/or ShoulderBaseInside can be used to define the actual start of the surface line. When either DikeToeAtPolder or ShoulderBaseInside is used , that point will be used as the actual dike toe for the calculations. If both types are used, ShoulderBaseInside is used. If neither is used, the first point of the surface line is considered to be the start point.
- DitchDikeSide, BottomDitchDikeSide, BottomDitchPolderSide, DitchPolderSide should be used to define a ditch. Note that when a ditch is defined, all four types must be used and in the correct order (as given, left to right).

The second complex type is the SoilSurface. This is a data object of the PipingProfile class. This class has the next input (properties) that have to be provided:

- Name (type = string): the name of the surface line.
- BottomLevel (type = double): the overall bottom level of the profile.
- Layers (type = List<PipingLayer>): a list of layers of type PipingLayer defining the profile. The layers must be ordered from top to bottom. For the definition of the PipingLayer, see below.

The class PipingLayer has the next properties:

- Name (type = string): the name of the layer.
- TopLevel (type = double): the top level of the layer
- AbovePhreaticLevel (type = double): the unit weight above the phreatic level of the soil/material in the layer.

- BelowPhreaticLevel (type = double): the unit weight below the phreatic level of the soil/material in the layer.
- DryUnitWeight (type = double) : the (oven) dry unit weight of the soil/material in the layer.
- IsAquifer (type = Boolean): indicates whether the layer is to be seen as aquifer layer (true) or not (false).

Note that a profile must have at least one aquifer layer and that each layer has a required minimum thickness of 0.001 m.

2.2.2 Determination of the exit point

To fulfil requirement 2 of the "Requirements and Functional Design", the class ExitPointDeterminator is created as part of the Deltares.WTIPiping assembly.

To be able to determine the exit point, the next input (properties) is required:

- PhiPolder: the piezo metric head at the side of the polder.
- RToe: the damping factor at the dike toe (optional, default value = 1).
- HRiver: the water level of the river.
- LeakageLenght: the leakage length.
- SurfaceLine: a line of type PipingSurfaceLine (see 2.2.1) defining the contour of the surface level of the dike including a possible ditch. For more information on the PipingSurfaceLine class and the PipingPoint class see also the Technical specifications (Lit. 3).
- SoilProfile: a one dimensional profile of type PipingProfile (see 2.2.1) containing all relevant layers and soil parameters per layer. It also holds the BottomLevel which defines the overall bottom level of the profile. For more information on the PipingProfile class and the PipingLayer class see also the Technical specifications (Lit. 3).
- RequiredFactorOfSafety: the required factor of safety for Uplift.
- VolumetricWeightOfWater: the volumetric weight of water (optional, default value = 9.81).
- ModelFactorUplift: the model factor for uplift.
- IsOvenDryUnitWeight: determines whether the oven dry unit weight should be used instead of the unit weight above phreatic level.
- MinimumThicknessCoverLayer: the minimum thickness of the cover layer.
- RExit: the damping factor at the exit point. This is an optional parameter with a default value of 1.

To be able to validate the input the Validate() function can be used. This function will report any errors found in a list of string. If no errors were found, the returned list will be empty. For this class the next conditions are checked during validation:

- SoilProfile is known (not null).
- SurfaceLine is known (not null).
- SurfaceLine contains minimal 2 points.
- All points in the surface line are ordered left to right.
- All characteristic points (if any) in the surface line are ordered left to right.
- The ditch (when used) is correct (all four ditch points are available and given in the correct left to right order).
- RExit is not 0.
- (PhiExit – Hexit) is not 0.
- SoilProfile has at least one layer
- The layers within the SoilProfile must be ordered top to bottom and must be at least 0.001 m apart.

- The bottomlevel of the SoilProfile must be at least 0.001 m is below the TopLevel of the deepest layer.
- SoilProfile has an aquifer.
- SoilProfile covers the surfaceline (top level SoilProfile at least as high as the surface line, bottom level SoilProfile at least as low as the surface line).
- SoilProfile covers the top of the aquifer (top level SoilProfile at least as high as the aquifer's top, bottom level SoilProfile at least as low as aquifer's top).

To start the determination, use the Calculate() function. After this, the property ExitPoint (type PipingPoint) can be used to obtain the resulting exit point.

2.2.3 Determination of the uplift margin.
To fulfil requirement 1 of the "Requirements and Functional Design", the class UpliftMarginDeterminator is created as part of the Deltares.WTIPiping assembly.

This class is inherited from the ExitPointDeterminator class and therefore has all the properties and functions defined for that class. Figure 1 shows the class diagram for this relation.

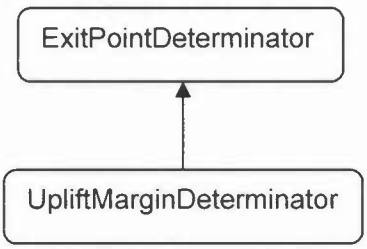


Figure 1: Class diagram of the Determinator classes.

The Validate() function offers exact the same checks as the base class ExitPointDeterminator.

In addition to the base class this class offers the next properties as result:
- PhiCu: an array of doubles containing all critical head values per X-coordinate.
- Phi: an array of doubles containing all calculated head values per X-coordinate.
- X: an array of doubles containing all X-coordinates.

2.2.4 Determination of the effective height and effective stress at exit point.
To help fullfill requirement 3, is should be possible to determine the effective stress and effective height. For this the class EffectiveThicknessCalculator is created as part of the Deltares.WTIPiping assembly. This is in fact a utility class, meant to assist in the determination of the effective stress and effective height for a given exit point. As this class is a utility class there is no validation. Furthermore its parameters are not part of Annex A as the results here are input for other functions.

To be able to determine the values, the next input (properties) is required:
- VolumetricWeightOfWater: the volumetric weight of water (optional, default value = 9.81).
- SurfaceLine: see 2.2.1 for description.

- SoilProfile: see 2.2.1 for description. Note that if the soil profile holds more than one aquifer, only the highest aquifer will be taken into account for the determination of the resulting values.
- ExitPointXCoordinate: the (local) x coordinate of the exit point.
- PhreaticLevel: the phreatic level at the side of the polder.

To start the determination, use the Calculate() function. After this, the next properties are available as results:

- EfffectiveStress: the calculated effective stress at the top of the aquifer at the given exit point x coordinate.
- EffectiveHeight: the calculated effective height above the aquifer at the given exit point x coordinate.

2.2.5 Determination of uplift safety.

To fulfil the uplift part of requirement 3 of the "Requirements and Functional Design", the class WTIUpliftCalculator is created as part of the Deltares.WTIPiping assembly.

To be able to determine the uplift safety, the next input (properties) is required:

- VolumetricWeightOfWater: the volumetric weight of water (optional, default value = 9.81).
- ModelFactorUplift: the model factor for uplift.
- EffectiveStress: the effective stress.
- HRiver: the water level of the river.
- PhiExit: the hydraulic head at exit point.
- RExit: the damping factor at the exit point. This is an optional parameter with a default value of 1.
- HExit: the phreatic level at exit point.
- PhiPolder: the piezo metric head at the side of the polder.

To be able to validate the input the Validate() function can be used. This function will report any errors found in a list of string. If no errors were found, the returned list will be empty. For this class the next conditions are checked during validation:

- RExit is not 0.
- (PhiExit – Hexit) is not 0.

To start the determination, use the Calculate() function. After this, the next properties are available as results:

- Zu: the limit state value for uplift.
- FoSu: the factor of safety for uplift.
- DeltaPhiCu: the critical head difference for uplift.
- Hcu: the critical water level for uplift.

Note that (PhiExit – Hexit) < 0 will result in a FoSu-value of 99.0. Also note, that when the EffectiveStress < 0 the kernel will raise an exception of type WTIUpliftCalculatorException.

2.2.6 Determination of piping safety (without cuttoff walls).

To fulfil the piping part of requirement 3 of the "Requirements and Functional Design", a number of classes are created as part of the Deltares.WTIPiping assembly. As the three piping models share common parameters and functions and both Sellmeijer models even more, using base classes and inheritance is appropriate. Figure 2 shows the class diagram for the piping models.

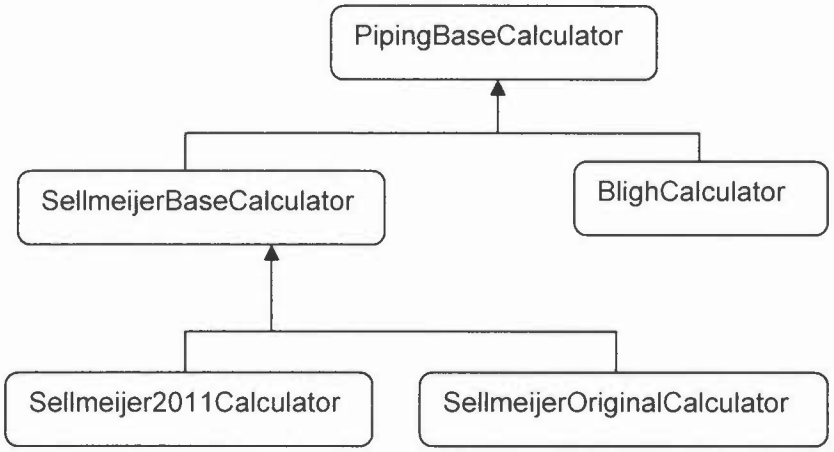


Figure 2: Class diagram Piping (without cuttoff walls) models

The class PipingBaseCalculator is the base class for all three piping calculation models that are to be implemented in accordance with requirement 4 (Sellmeijer – single layer - revised, Sellmeijer – single layer – original and Bligh). This base class contains all common parameters and functions as shared by the mentioned piping models.

Its input properties are:

- ModelFactorPiping: the model factor for piping.
- HRiver: the river level.
- HExit: the phreatic level at exit point.
- Rc: the reduction factor providing the fraction of the blanket thickness by which the total head difference is reduced due to the hydraulic resistance in the vertical exit channels (optional, default value = 0.3).
- DTotal: the total thickness of the cover layer.
- SeepageLength: the seepage length.

The functions Validate() and Calculate() are both implemented in the base class.
For this class the next condition is checked during validation:

- (HRiver – Hexit - (Rc * DTotal)) is not 0.

The output properties are:

- Zp: limit state function value for piping.
- FoSp: safety factor for piping.
- Hcp: the critical water level for piping.
- Hc: the critical head difference for piping.

The SellmeijerBaseCalculator is the base class for both Sellmeijer models. It extends the PipingBaseCalculator with several additional input properties:

- GammaSubParticles: the submerged volumetric weight of the sand particles (optional, default value = 16.5).
- WhitesDragCoefficient the whites drag coefficient (optional, default value = 0.25).
- D70: 70% fractile of the aquifer's grain size distribution.
- VolumetricWeightOfWater: volumetric weight of water (optional, default value = 9.81).
- DarcyPermeability: the permeability (Darcy).
- KinematicViscosityWater: the kinematic viscosity of water (optional, default value = 1.33 e-6).
- Gravity: the gravitational constant (optional, default value = 9.81).
- DAquifer: the thickness of the aquifer.

The Validate() function offers the next additional checks to base class:

- VolumetricWeightOfWater is not 0.
- SeepageLength is greater than 0.
- DAquifer is greater than 0.

Piping safety using Sellmeijer – single layer – revised

To fulfil the piping part of requirement 3 and requirement 4a of the "Requirements and Functional Design", the class Sellmeijer2011Calculator is created as part of the Deltares.WTIPiping assembly.

The Sellmeijer2011Calculator is inherited from the base class SellmeijerBaseCalculator and extends this with additional input properties:

- D70Mean: the reference value for Sellmeijer of the 70% fractile of the aquifer's grain size distribution (optional, default value = 2.08 e-4).
- BeddingAngle: the bedding angle for the Sellmeijer revised model (optional, default value = 37).

The Validate() function offers the next additional checks to base class:

- BeddingAngle is at least 0.

Piping safety using Sellmeijer – single layer – original

To fulfil the piping part of requirement 3 and requirement 4b of the "Requirements and Functional Design", the class SellmeijerOriginalCalculator is created as part of the Deltares.WTIPiping assembly.

The SellmeijerOriginalCalculator is inherited from the base class SellmeijerBaseCalculator and extends this with one additional input property:

- BeddingAngle: the bedding angle for the Sellmeijer original model (optional, default value = 41).

The Validate() function offers the next additional checks to base class:

- BeddingAngle is at least 0.
- WhitesDragCoefficient is greater than 0.

Piping safety using Bligh

To fulfil the piping part of requirement 3 and requirement 4c of the "Requirements and Functional Design", the class BighCalculator is created as part of the Deltares.WTIPiping assembly.

The BighCalculator is inherited from the base class PipingBaseCalculator and only extends this with one additional input property:

- D50: the median diameter of the aquifer's grain size distribution.

2.2.7 Determination of the piping safety (with cuttoff walls) using Lane.

To fulfil requirement 5 of the "Requirements and Functional Design", the class LaneCalculator is created as part of the Deltares.WTIPiping assembly.

To be able to determine the piping safety using Lane, the next input (properties) is required:

- LHorizontal: the horizontal seepage length.
- LVertical: the vertical seepage length.
- HRiver: the water level of the river.
- HExit: the phreatic level at exit point.
- D50: the median diameter of the aquifer's grain size distribution.

To be able to validate the input the Validate() function can be used. This function will report any errors found in a list of string. If no errors were found, the returned list will be empty. For this class the next condition is checked during validation:

- (HRiver – Hexit) is not 0.

To start the determination, use the Calculate() function. After this, the next properties are available as results:

- ZLane: the limit state value for Lane.
- FoSLane: the factor of safety for Lane.
- HcLane: the critical water level for Lane.

2.2.8 Determination of heave safety (without cuttoff walls).

To fulfil the heave part of requirement 3 and requirement 6 of the "Requirements and Functional Design", the class HeaveCalculator is created as part of the Deltares.WTIPiping assembly.

To be able to determine the heave safety, the next input (properties) is required:

- Ich: the critical heave gradient.
- DTotal: the total thickness of the cover layer.
- PhiExit: the hydraulic head at exit point.
- RExit: the damping factor at the exit point. This is an optional parameter with a default value of 1.
- HExit: the phreatic level at exit point.
- PhiPolder: the piezo metric head at the side of the polder.

To be able to validate the input the Validate() function can be used. This function will report any errors found in a list of string. If no errors were found, the returned list will be empty. For this class the next condition is checked during validation:

- DTotal is not 0.
- RExit is not 0.
- (PhiExit – Hexit) is not 0.

To start the determination, use the Calculate() function. After this, the next properties are available as results:

- Zh: the limit state value for heave.
- FoSh: the factor of safety for heave.
- Hch: the critical water level for heave.

Note that (PhiExit – Hexit) < 0 will result in a FoSh-value of 99.0.

2.2.9 Determination of heave safety (with cuttoff walls).

To fulfil the heave part of requirement 3 of the "Requirements and Functional Design", the class HeaveCuttOffCalculator is created as part of the Deltares.WTIPiping assembly.

This class is inherited from the HeaveCalculator class and therefore has all the properties and functions defined for that class. Figure 3 shows the class diagram for this relation.

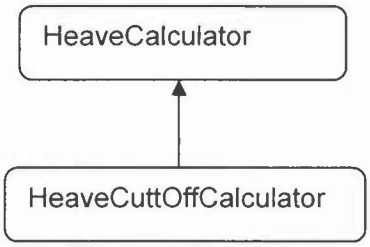


Figure 3: Class diagram of the Heave classes.

In addition to the base class this class requires one extra property as input:

- Gradient: the exit gradient.

The Validate() function is redefined instead of inherited and checks:

- RExit is not 0.
- Gradient is not 0.

3 Literature

1. WTI guidelines for software development incl. audit checklist
   https://repos.deltares.nl/repos/Ringtoets/trunk/doc/Guidelines, templates and checklists/WTI guidelines for software development incl audit checklist.doc
2. RTO design.doc
   https://repos.deltares.nl/repos/Ringtoets/trunk/doc/system/RTO design.doc
3. Technical specifications
   https://repos.deltares.nl/repos/FailureMechanisms/FailureMechanisms/DikesPiping/trunk/doc/DoxyGen/API Docs/latex/Piping Kernel - Technical Documentation.pdf

A Full description of parameters

The next table shows the full description of all input parameters.

Parameter name	Description	Unit	Type and precision. In case of a list: single or multiple values	Default value	Min value	Max value
HRiver	Water level of the river	[m] reference level	Real number, 0.000	*None*	-1 * Max Double	Max Double
HExit	Phreatic r level at exit point	[m] reference level	Real number, 0.000	*None*	-1 * Max Double	Max Double
PhiPolder	Piezo metric head at the side of the polder	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
PhiExit	Piezo metric head at exit point	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
RToe	Damping factor at dike toe	[-]	Real number, 0.000	1	0	1
RExit	Damping factor at exit point	[-]	Real number, 0.000	1	0	1
LeakageLength	Leakage length	[m]	Real number, 0.000	*None*	0	Max Double
VolumetricWeightOfWater	Volumetric weight of water	[kN/m^3]	Real number, 0.000	9.81	-1 * Max Double	Max Double
DAquifer	Thickness of the aquifer	[m]	Real number, 0.000	*None*	>0	Max Double
RequiredFactorOfSafety	Required factor of safety for uplift	[-]	Real number, 0.000	*None*	-1 * Max Double	Max Double
ModelFactorUplift	Model factor for uplift	[-]	Real number, 0.000	*None*	-1 * Max Double	Max Double
IsOvenDryUnitWeight	Use oven dry unit weight instead of unit weight above phreatic level?	[-]	Boolean	*None*	*Inapp.*	*Inapp.*
MinimumThicknesCoverLayer	Minimum thickness of the cover layer	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
EffectiveStress	Effective stress	[kN/m^3]	Real number, 0.000	*None*	-1 * Max Double	Max Double

ModelFactorPiping	Model factor for piping	[-]	Real number, 0.000	*None*	-1 * Max Double	Max Double
Rc	Reduction factor blanket thickness	[-]	Real number, 0.000	0.3	-1 * Max Double	Max Double
DTotal	Total thickness of cover layer	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
SeepageLength	Seepage length	[m]	Real number, 0.000	*None*	>0	Max Double
GammaSubParticles	Submerged volumetric weight of sand particles	[kN/m^3]	Real number, 0.000	16.5	-1 * Max Double	Max Double
WhitesDragCoefficient	Whites drag coefficient	[-]	Real number, 0.000	0.25	>0	Max Double
D70	70% fractile of the aquifers grain size distribution	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
DarcyPermeability	Permeability (Darcy)	[m/s]	Real number, 0.000	*None*	-1 * Max Double	Max Double
KinematicViscosityWater	Kinematic viscosity of water	[m^2/s]	Real number, 0.000	1.33E-6	-1 * Max Double	Max Double
Gravity	Gravitational constant	[m/s^2]	Real number, 0.000	9.81	-1 * Max Double	Max Double
D70Mean	Reference value for Sellemeijer	[m]	Real number, 0.000	*2.08E-4*	-1 * Max Double	Max Double
BeddingAngle (revised)	Bedding angle for the Sellemijer revised model	[°]	Real number, 0.000	37	0	Max Double
BeddingAngle (Original)	Bedding angle for the Sellmeijer original model	[°]	Real number, 0.000	41	0	Max Double
D50	Median diameter of the aquifer's grain size distribution	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
LHorizontal	Horizontal seepage length	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
LVertical	Vertical seepage length	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
Ich	Critical	[-]	Real number, 0.000	*None*	-1 *	Max

	heave gradient				Max Double	Double
SurfaceLine	Line of points defining the surface level, see 2.2.1	*Inapp.*	PipingSurfaceLine	*None*	*Inapp.*	*Inapp.*
PipingSurfaceLine.Name	Name of the surface line	*Inapp.*	String	*None*	*Inapp.*	*Inapp.*
PipingSurfaceLine.Points	Points defining the surface line	*Inapp.*	List<PipingPoint>	*None*	*Inapp.*	*Inapp.*
PipingPoint	Point with all relevant properties	*Inapp.*	PipingPoint	*None*	*Inapp.*	*Inapp.*
PipingPoint.X	x-coordinate	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
PipingPoint.Y	y-coordinate	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
PipingPoint.Z	z-coordinate	[m]	Real number, 0.000	*None*	-1 * Max Double	Max Double
PipingPoints.Type	Type	*Inapp.*	PipingCharacteristic PointType	None		
SoilProfile	1D profile defining the layers and their parameters, see 2.2.1	*Inapp.*	PipingProfile	*None*	*Inapp.*	*Inapp.*
SoilProfile.Name	Name of the profile	*Inapp.*	String	*None*	*Inapp.*	*Inapp.*
SoilProfile.BottomLevel	Bottom level	[m] reference level	Real number, 0.000	*None*	-1 * Max Double	Max Double
SoilProfile.Layers	Layers defining the profile	*Inapp.*	List<PipingLayer>	*None*	*Inapp.*	*Inapp.*
PipingLayer	Layer with all relevant properties	*Inapp.*	PipingLayer	*None*	*Inapp.*	*Inapp.*
PipingLayer.Name	Name of the surface line	*Inapp.*	String	*None*	*Inapp.*	*Inapp.*
PipingLayer.TopLevel	Top level of the layer	[m] reference level	Real number, 0.000	*None*	-1 * Max Double	Max Double
PipingLayer.AbovePhreaticLevel	Unit weight above phreatic level	[kN/m^3]	Real number, 0.000	*None*	-1 * Max Double	Max Double
PipingLayer.BelowPhreaticLevel	Unit weight below phreatic level	[kN/m^3]	Real number, 0.000	*None*	-1 * Max Double	Max Double

PipingLayer.DryUnitWeight	(Oven) Dry Unit weight	[kN/m^3]	Real number, 0.000	*None*	-1 * Max Double	Max Double
PipingLayer.IsAquifer	Aquifer indicator	*Inapp.*	Boolean	False	*Inapp.*	*Inapp.*

The next table shows the full description of all output parameters.

Parameter name	Description	Unit	Type and precision. In case of a list: single or multiple values	Default value	Min value	Max value
ExitPoint	Exit Point (as GeometryPoint)	*Inapp.*	GeometryPoint	*None*	*Inapp.*	*Inapp.*
PhiCu	Critical head values per X coordinate	[m]	Array of Real numbers	*None*	*Inapp.*	*Inapp.*
Phi	Calculated head values per X coordinate	[m]	Array of Real numbers	*None*	*Inapp.*	*Inapp.*
X	X coordinates	[m]	Array of Real numbers	*None*	*Inapp.*	*Inapp.*
Zu	Limit state value for uplift	[-]	Real number	*None*	*Inapp.*	*Inapp.*
FoSu	Factor of safety for uplift	[-]	Real number	*None*	*Inapp.*	*Inapp.*
DeltaPhiCu	Critical head difference for uplift	[m]	Real number	*None*	*Inapp.*	*Inapp.*
Hcu	Critical water level for uplift	[m] reference level	Real number	*None*	*Inapp.*	*Inapp.*
Zp	Limit state value for piping	[-]	Real number	*None*	*Inapp.*	*Inapp.*
FoSp	Factor of safety for piping	[-]	Real number	*None*	*Inapp.*	*Inapp.*
Hcp	Critical water level for piping	[m] reference level	Real number	*None*	*Inapp.*	*Inapp.*
Hc	Critical head difference for piping	[m]	Real number	*None*	*Inapp.*	*Inapp.*
ZLane	Limit state value for Lane	[-]	Real number	*None*	*Inapp.*	*Inapp.*
FoSLane	Factor of safety for Lane	[-]	Real number	*None*	*Inapp.*	*Inapp.*
HcLane	Critical water level for Lane	[m] reference level	Real number	*None*	*Inapp.*	*Inapp.*
Zh	Limit state value for Heave	[-]	Real number	*None*	*Inapp.*	*Inapp.*
FoSh	Factor of safety for Heave	[-]	Real number	*None*	*Inapp.*	*Inapp.*
Hch	Critical water level for Heave	[m] reference level	Real number	*None*	*Inapp.*	*Inapp.*

B Examples of implementation

In this annex an example of the implementation is provided for each of the required functions. These examples show (in C#) what type of object are to be created, how to set their properties, how to start the calculation and how to obtain the results.

B.1 Determination of the exit point

First the surface line has to be created and filled. This can be done in a function like:

```
public PipingSurfaceLine CreateSurfaceLineForExitPointTest()
{
	// create a surface line
	PipingSurfaceLine surfaceLine = new PipingSurfaceLine { Name =
	"ExitPointLine"};
	// add a point as dike toe
	surfaceLine.Points.Add(new PipingPoint(10.0, 0, 0,
	PipingCharacteristicPointType.DikeToeAtRiver));

	// add point as top of ditch at dike side
	surfaceLine.Points.Add(new PipingPoint(58.5, 0, 0,
	PipingCharacteristicPointType.DitchDikeSide));

	// add point as bottom of ditch at dike side
	surfaceLine.Points.Add(new PipingPoint(59.5, 0, -2,
	PipingCharacteristicPointType.BottomDitchDikeSide));

	// add point as bottom of ditch at polder side
	surfaceLine.Points.Add(new PipingPoint(61.5, 0, -2,
	PipingCharacteristicPointType.BottomDitchPolderSide));

	// add point as top of ditch at polder side
	surfaceLine.Points.Add(new PipingPoint(61.5, 0, 0,
	PipingCharacteristicPointType.DitchPolderSide));

	// add non typed point
	surfaceLine.Points.Add(new PipingPoint(70.0, 0, -0.2));

	// add point as end point of the surface line
	surfaceLine.Points.Add(new PipingPoint(75.0, 0, 0,
	PipingCharacteristicPointType.SurfaceLevelInside));

	// give the now filled surface line as result
	return surfaceLine;
}
```

Next the soil profile, together with its layers and their parameters, has to be created and filled. This can be done in a function like:

```
public static PipingProfile CreateSimpleTestProfile()
{
	// create the profile, make sure to set the BottomLevel!
	var soilProfile = new PipingProfile
	{
		Name = "TestProf",
		BottomLevel = -30
```

```
        };
        // create a layer, set the paramters and add it to the profile
        var layer1 = new PipingLayer
        {
                Name = "layer1 - klei",
                AbovePhreaticLevel = 18,
                BelowPhreaticLevel = 20,
                TopLevel = 10,
                IsAquifer = false
        };
        soilProfile.Layers.Add(layer1);
        // create another layer, set the paramters and add it to the profile
        var layer2 = new PipingLayer
        {
                Name = "layer2 - zand",
                AbovePhreaticLevel = 15,
                BelowPhreaticLevel = 17,
                TopLevel = -3,
                IsAquifer = true
        };
        soilProfile.Layers.Add(layer2);

        // give the now filled soil profile as result
        return soilProfile;
}
```

Now create and fill the determinator object:

```
public ExitPointDeterminator FillSimpleExitPointDeterminator()
{
        // create the detrminator and set all required properties
        var epCalc = new ExitPointDeterminator();
        epCalc.SoilProfile = CreateSimpleTestProfile();
        epCalc.SurfaceLine = CreateSurfaceLineForExitPointTest();
        epCalc.PhiPolder = 0;
        epCalc.HRiver = 4;
        epCalc.IsUseOvenDryUnitWeight = false;
        epCalc.LeakageLength = 0;
        epCalc.MinimumThicknessCoverLayer = 0;
        epCalc.ModelFactorUplift = 1;
        epCalc.RExit = 1;
        epCalc.RToe = 1;
        epCalc.RequiredFactorOfSafety = 1;
        epCalc.VolumetricWeightOfWater = 10;
        epCalc.LeakageLength = 30;
        return epCalc;
}
```

Use the determinator to validate the data:

```
public void ExampleValidate()
{
	// create and fill the determinator
	var epCalc = FillSimpleExitPointDeterminator();


	// start the validation
	var errors = epCalc.Validate();
	// check the errors, the count tells how much there are (0 if none)
	bool valid = true;
	if (errors.Count > 0)
	{
		valid = false;
		string firstError = errors[0];
	}
}
```

The Validate() example above is in fact a generic example of how to implement this method for all the required functions (use the object.Validate() to get a list of errors).

Use the determinator to get results:

```
public void ExampleCalculate()
{
	// create and fill the determinator
	var epCalc = FillSimpleExitPointDeterminator();

	// start the calculation
	epCalc.Calculate();
	// retrieve the result
	var myExitPoint = epCalc.ExitPoint;
	// check to see if it is null (it is null when no uplift occurs and
	// no point can be found. If available then the coordinates of the
	// exit point can be retrieved.
	if (myExitPoints != null)
	{
		var myX = epCalc.ExitPoint.X;
		var myZ = epCalc.ExitPoint.Z;
	}
}
```

B.2 Determination of the uplift margin

Create and fill the determinator object (using the surface line and soilprofile as the example for the exit point):

```
public UpliftMarginDeterminator FillSimpleUpliftMarginDeterminator()
{
    var umdCalc = new UpliftMarginDeterminator();
    umdCalc.SoilProfile = CreateSimpleTestProfile();
    umdCalc.SurfaceLine = CreateSurfaceLineForExitPointTest();
    umdCalc.PhiPolder = 0;
    umdCalc.HRiver = 4;
    umdCalc.IsUseOvenDryUnitWeight = false;
    umdCalc.LeakageLength = 0;
    umdCalc.MinimumThicknessCoverLayer = 0;
```

```
            umdCalc.ModelFactorUplift = 1;
            umdCalc.RExit = 1;
            umdCalc.RToe = 1;
            umdCalc.RequiredFactorOfSafety = 1;
            umdCalc.VolumetricWeightOfWater = 10;
// Make leekage lenght very long so PhiAtX returns PhiToe (or near that).
            umdCalc.LeakageLength = 3000000;
            return umdCalc;
        }
```

Use the determinator to get results:

```
        public void ExampleCalculate()
        {
            const double CDiff = 0.001;
            var umdCalc = FillSimpleUpliftMarginDeterminator();

            umdCalc.Calculate();
            Assert.AreEqual(50.5, umdCalc.X[0], CDiff);
            Assert.AreEqual(4.0, umdCalc.Phi[0], CDiff);
            Assert.AreEqual(3.0, umdCalc.PhiCu[0], CDiff);
            Assert.AreEqual(60.5, umdCalc.X[20], CDiff);
            Assert.AreEqual(4.0, umdCalc.Phi[20], CDiff);
            Assert.AreEqual(1.0, umdCalc.PhiCu[20], CDiff);
            Assert.AreEqual(75, umdCalc.X[49], CDiff);
            Assert.AreEqual(4.0, umdCalc.Phi[49], CDiff);
            Assert.AreEqual(3.0, umdCalc.PhiCu[49], CDiff);
        }
```

B.3 Determination of effective height and effective stress at exit point

```
        public void ExampleCalculate()
        {
            var effectiveThicknessCalculator = new
               EffectiveThicknessCalculator
            {
                SurfaceLine = CreateSurfaceLineForExitPointTest(),
                SoilProfile = CreateSimpleTestProfile(),
                PhreaticLevel = -1,
                VolumicWeightOfWater = 10
            };
            // Set the top of the aquifer to the top of the second layer (-3)
            effectiveThicknessCalculator.ExitPointXCoordinate = 56; // left of
            //  the ditch
            effectiveThicknessCalculator.Calculate();
            // Top of bottom aquifer is -3
            // Surfacelevel = 0
            // Outside ditch, so effective height = surfacelevel -
            // top_bottom_aquifer = 10.0
            Assert.AreEqual(10.0,
               effectiveThicknessCalculator.EffectiveHeight, 0.001);

        }
```

B.4 Determination of the uplift safety

```
        public void ExampleUplift()
        {
// create and fill the calculator
```

```
            var upliftCalculator = new WTIUpliftCalculator();
            upliftCalculator.VolumetricWeightOfWater = 10;
            upliftCalculator.ModelFactorUplift = 1.0;
            upliftCalculator.EffectiveStress = 45;
            upliftCalculator.RExit = 0.8;
            upliftCalculator.HRiver = 3.0;
            upliftCalculator.HExit = 0.6;
            upliftCalculator.PhiPolder = 0.5;
// local use of a special function to get a good value for PhiExit
            upliftCalculator.PhiExit =
PiezoHeadCalculator.CalculatePhiExit(upliftCalculator.PhiPolder,
                upliftCalculator.RExit, upliftCalculator.HRiver);
            upliftCalculator.Calculate();
            // PhiExit = PhiPolder + RExit * (HRiver - PhiPolder) = 0.5 + 0.8
* (3.0 - 0.5) = 2.5
            // DelptaPhiCu = EffectiveStress / VolumetricWeightOfWater = 45 /
10 = 4.5
            Assert.AreEqual(4.5, upliftCalculator.DeltaPhiCu, 0.0001);
            // Zu = ModelFactorUplift * DelptaPhiCu - (PhiExit - HExit) = 1.0
* 4.5 - (2.5 - 0.6) = 2.6
            Assert.AreEqual(2.6, upliftCalculator.Zu, 0.0001);
            // FosU = ModelFactorUplift * DelptaPhiCu / (PhiExit - HExit) =
1.0 * 4.5 / (2.5 - 0.6) = 2.6
            Assert.AreEqual(2.368421, upliftCalculator.FoSu, 0.0001);
            // Hcu = (DelptaPhiCu + HExit - PhiPolder) / RExit + PhiPolder =
(4.5 + 0.6 - 0.5) / 0.8 + 0.5
            Assert.AreEqual(6.25, upliftCalculator.Hcu, 0.0001);
        }
```

B.5 Determination of the piping safety using Sellmeijer revised

First, set up the calculator:

```
        public Sellmeijer2011Calculator SetUpSellmeijerRevisedCalculator()
        {
            var sc = new Sellmeijer2011Calculator();
            sc.WhitesDragCoefficient = 2.0;
            sc.BeddingAngle = 45;
            sc.GammaSubParticles = 4000;
            sc.VolumetricWeightOfWater = 1000;
            sc.D70 = 2.0;
            sc.SeepageLength = 30.0;
            sc.DarcyPermeability = 0.0001;
            sc.DAquifer = 3.0;
            return sc;
        }
```

Then use it to get results:

```
        public void ExampleCalculate()
        {
            var sc = SetUpSellmeijerRevisedCalculator();
            sc.HRiver = 1;
            sc.Calculate();
            double expectedResult = 139.33425195 * 30;
            Assert.AreEqual(expectedResult, sc.Hc, 0.0001);
            Assert.AreEqual(-1, sc.Zp, 0.0001);
            Assert.AreEqual(0, sc.FoSp, 0.0001);
```

```
            Assert.AreEqual(0, sc.Hcp, 0.0001);
        }
```

B.6 Determination of the piping safety using Sellmeijer original

```
        public void ExampleCalculate()
        {
            var sc = new SellmeijerOriginalCalculator();
            sc.ModelFactorPiping = 0.996815279;
            sc.WhitesDragCoefficient = 0.298511157;
            sc.BeddingAngle = 42.8957293359575;
            sc.GammaSubParticles = 17;
            sc.VolumetricWeightOfWater = 10;
            sc.D70 = 0.000224485988029883;
            sc.SeepageLength = 92.10573174;
            sc.DarcyPermeability = 0.000297428;
            sc.DAquifer = 15.32373435;
            sc.DTotal = 8.001735065;
            sc.KinematicViscosityWater = 0.00000133;

            sc.HRiver = 3.123489063;
            sc.HExit = 1.5;
            sc.Calculate();
            // Result of Zp with default value for Rc = 0.3
            // Zp = mp * Hc - (hriver - hexit - rc * d)
            //    = 1.1 * sc.Hc - (3.123489063 - 1.5 - 0.3 * 8.001735065)
            //    = 1.1 * sc.Hc - 1.383489;
            double expectedResult = sc.ModelFactorPiping * sc.Hc - (-
0.777031457);
            double actualResult = sc.Zp;
            double expectedHc = 10.374078197166554;
            Assert.AreEqual(expectedHc, sc.Hc, 0.001);
            Assert.AreEqual(expectedResult, actualResult, 0.001);

            // Result of Zp for Rc = 0.5
            // Zp = mp * Hc - (hriver - hexit - rc * d)
            //    = 1.1 * sc.Hc - (3.123489063 - 1.5 - 0.5 * 8.001735065)
            //    = 1.1 * sc.Hc - -2.37737847;
            sc.Rc = 0.5;
            sc.Calculate();
            expectedResult = sc.ModelFactorPiping * sc.Hc - -2.37737847;
            actualResult = sc.Zp;
            Assert.AreEqual(expectedResult, actualResult, 0.001);
        }
```

B.7 Determination of the piping safety using Bligh

```
        public void ExampleCalculate()
        {
            const double diff = 0.0001;
            var bc = new BlighCalculator();
            bc.HRiver = 1;
            bc.Calculate();
            Assert.AreEqual(-1, bc.Zp);
            Assert.AreEqual(0, bc.FoSp);
            Assert.AreEqual(0, bc.Hcp);
            Assert.AreEqual(0, bc.Hc);
```

```
    bc.HRiver = 7;
    bc.ModelFactorPiping = 0.5;
    bc.HExit = 2;
    bc.Rc = 0.8;
    bc.DTotal = 3;
    bc.SeepageLength = 18;
    bc.D50 = 2.5e-3;
    bc.Calculate();
    Assert.AreEqual(-1.6, bc.Zp, diff);
    Assert.AreEqual(1/2.6, bc.FoSp, diff);
    Assert.AreEqual(5.4, bc.Hcp, diff);
    Assert.AreEqual(2, bc.Hc, diff);
}
```

B.8 Determination of the piping safety using Lane

```
public void ExampleCalculate()
{
    const double diff = 0.0001;
    var laneCalculator = new LaneCalculator();
    laneCalculator.HRiver = 1;
    laneCalculator.Calculate();
    Assert.AreEqual(0, laneCalculator.FoSLane, diff);
    Assert.AreEqual(-1, laneCalculator.ZLane, diff);
    Assert.AreEqual(0, laneCalculator.HcLane, diff);

    laneCalculator.HRiver = 7;
    laneCalculator.LHorizontal = 60;
    laneCalculator.LVertical = 10;
    laneCalculator.HExit = 2;
    laneCalculator.D50 = 2.5e-4;
    laneCalculator.Calculate();
    Assert.AreEqual(1, laneCalculator.FoSLane, diff);
    Assert.AreEqual(0, laneCalculator.ZLane, diff);
    Assert.AreEqual(7, laneCalculator.HcLane, diff);
}
```

B.9 Determination of the heave safety (without cuttoff walls)

```
public void ExampleCalculate()
{
    var hc = new HeaveCalculator();
    hc.PhiExit = 1;
    hc.RExit = 1;
    hc.DTotal = 1;
    hc.Calculate();
    Assert.AreEqual(-1, hc.Zh);
    Assert.AreEqual(0, hc.FoSh);
    Assert.AreEqual(0, hc.Hch);
    hc.PhiExit = 3;
    hc.RExit = 0.5;
    hc.DTotal = 2;
    hc.HExit = -1;
    hc.Ich = 2;
    hc.PhiPolder = -1;
    hc.Calculate();
    Assert.AreEqual(0, hc.Zh);
    Assert.AreEqual(1, hc.FoSh);
```

```
    Assert.AreEqual(7, hc.Hch);
}
```

B.10 Determination of the heave safety (with cuttoff walls)

```
public void ExampleCalculate()
{
    var hc = new HeaveCutoffCalculator();
    hc.Gradient = 1;
    hc.RExit = 1;
    hc.DTotal = 1;
    hc.Calculate();
    Assert.AreEqual(-1, hc.Zh);
    Assert.AreEqual(0, hc.FoSh);
    Assert.AreEqual(0, hc.Hch);
    hc.Gradient = 4;
    hc.RExit = 0.5;
    hc.DTotal = 2;
    hc.HExit = -1;
    hc.Ich = 2;
    hc.PhiPolder = -1;
    hc.Calculate();
    Assert.AreEqual(-2, hc.Zh);
    Assert.AreEqual(0.5, hc.FoSh);
    Assert.AreEqual(7, hc.Hch);
}
```