Deltares

Enabling Delta Life

DAM Engine

Technical Design

Deltares

DAM Engine

Technical Design

1210702-000

Deltares

Title
DAM Engine

Client	Project	Reference	Pages
Deltares - Geo engineering DKS	1210702-000	1210702-000-GEO-0004	41

Classification
-

Keywords
Dike, safety assessment, design, software, macro stability, piping

Summary
This document contains the technical design for DAM Engine, a software module that computes the strength of a complete dikering with respect to several failure mechanisms, such as macro stability and piping.

Samenvatting
Dit document bevat het technisch ontwerp voor DAM Engine, een software module die een gebruiker in staat stelt om voor een dijktraject berekeningen uit te voeren voor verschillende faalmechanismen, waaronder macrostabiliteit en piping.

References
Refer to chapter 8.

Version	Date	Author	Initials	Review	Initials	Approval	Initials
0.3	Mar 2017	Tom The		John Bokma André Grijze		Maya Sule	

Status
draft
This is a draft report, intended for discussion purposes only. No part of this report may be relied upon by either principals or third parties.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and scope of this document

This document contains the technical design for the DAM Engine, a computational engine for the automated calculation of the strength of dikes. DAM was developed by Deltares with and for STOWA for all water authorities. This document describes the full intended architecture of the DAM Engine. What will actually will be implemented depends on the requirements of the clients using this DAM Engine. If some functionality is not (yet) needed, then that part does not need to be implemented.

1.1.1 Future options

As mentioned above this document contains some options that will not be implemented in the first release, but are foreseen to be implemented in the near future. Therefore although sometimes a reference will be made to these options, these will not be described in detail yet.

That applies in particular to the following subjects:

- NWO module("Niet Waterkerende Objecten")
- WBI failure mechanisms (Piping, Macrostability)

1.2 Other system documents

The full documentation on the program comprises the following documents.

Title	Content
DAM Engine- Architecture Overall (The, 2017a)	Description of overall architecture of the DAM Engine and its components.
DAM Engine- Functional Design (Zwan, 2017)	Description of the requirements and functional design.
DAM Engine- Technical Design (this document) (The, 2017b)	Description of the implementation of the technical design of DAM Engine.
DAM Engine- Technical documentation (Doxygen, 2017)	Description of the arguments and usage of different software components, generated from in-line comment with Doxygen.
DAM Engine- Test Plan (Trompille, 2017a)	Description of the different regression and acceptation tests, including target values.
DAM Engine- Test Report (Trompille, 2017b)	Description of the test results (benchmarks and test scripts).
Architecture Guidelines (Kleijn *et al.*, 2017)	Architecture guidelines that are used by DSC-Deltares.

Table 1.1: *DAM Engine system documents.*

1.3 Document revisions

1.4 Document revisions

1.4.1 Revision 0.1

First concept of the document.

1.4.2 Revision 0.2

Adapted based on reviews of this document by Jan Noort and André Grijze.

1.4.3 Revision 0.3

Adapted based on review of this document by John Bokma.

2 System Architecture

This chapter contains diagrams describing the modules and submodules of the DAM Engine and how they interact. In chapter 6 a short description of each module/submodules is given.

2.1 DAM components

DAM Engine is part of the whole DAM system that contains several components. Please see Figure 2.1 for an overview of the components of DAM. In (The, 2017a) a description of the overall architecture of the DAM system can be found.

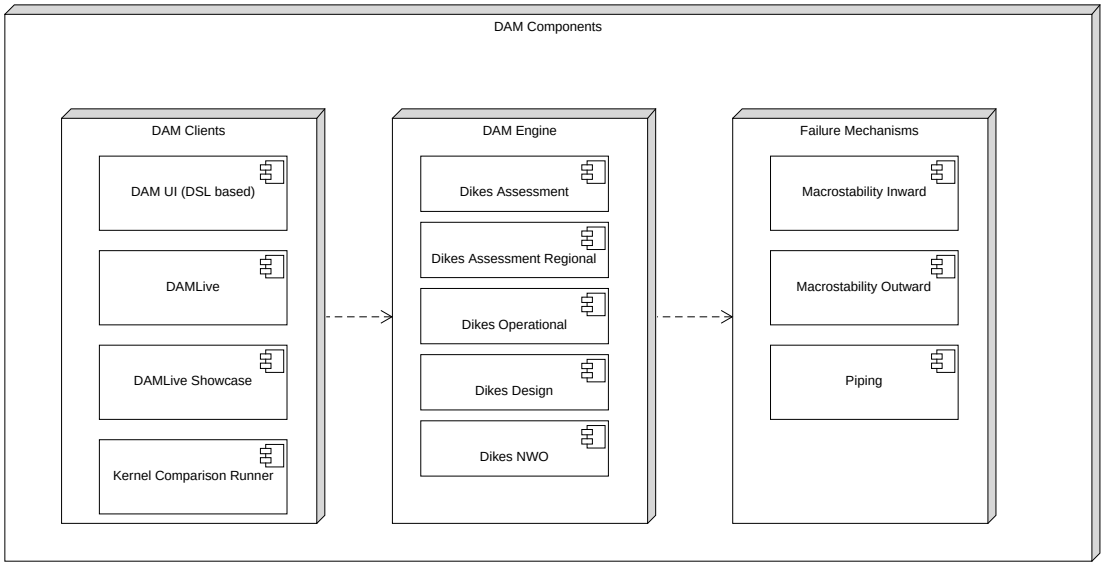

Figure 2.1: *DAM Engine and its components.*

The arrows illustrate the dependencies of the components.

2.2 DAM Engine data flow

Please see Figure 2.2 for an overview of the data flow within the DAM system.

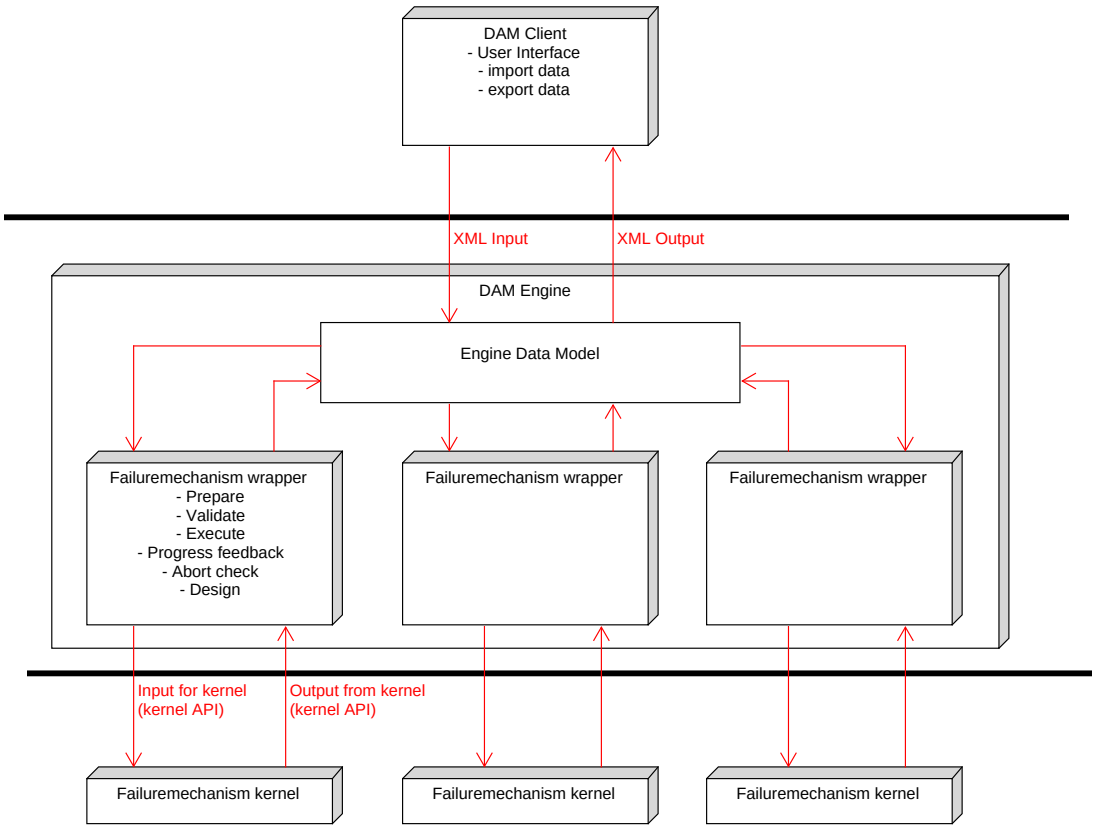

Figure 2.2: *DAM Engine and its components.*

The red arrows illustrate the dataflow between the main DAM components.
As can be seen the data exchange between the DAM Engine and the failuremechanism kernel (bottom of the picture) is done through the API that is defined by the failuremechanism kernel. The data exchange between the DAM Engine and the DAM client (top of the picture) is done through XML files (one for input and one for output), which are well defined by XML schemas (XSD's).

2.3 DAM Engine components

The DAM Engine itself also consists of several modules. These can be seen in see Figure 2.3

All of the submodules inside the Main Modules are completely independent. All of the submodules inside the Supporting Modules are also independent. But all these submodules can be used by each of the main modules. The arrows show the allowed dependencies.

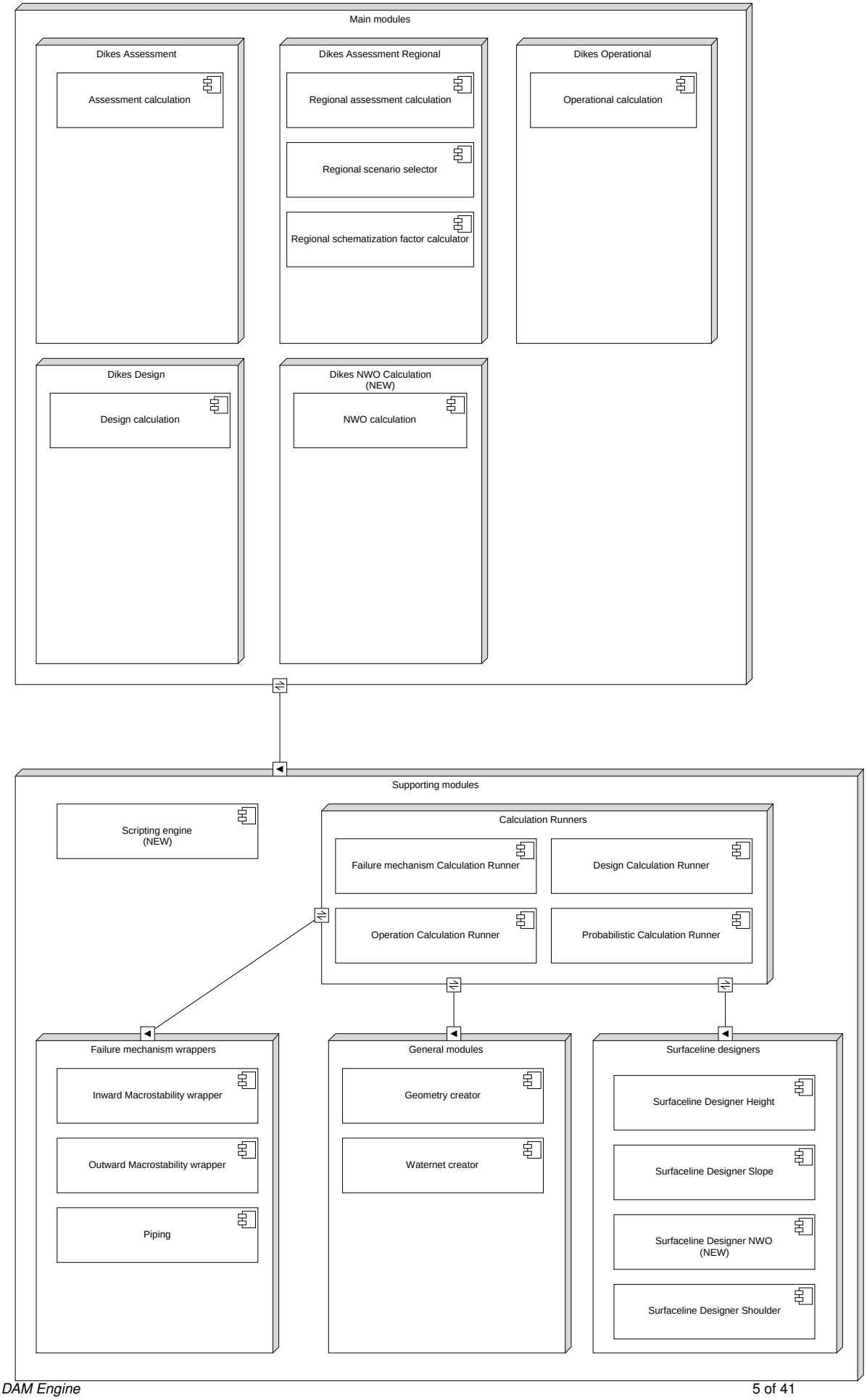

Figure 2.3: *DAM Engine and its components.*

2.4 DAM Engine sequence and activity diagrams

In this section the sequence diagrams, showing the use of the submodules are shown. For each sequence diagram a corresponding activity diagram is also shown

2.4.1 Dikes assessment

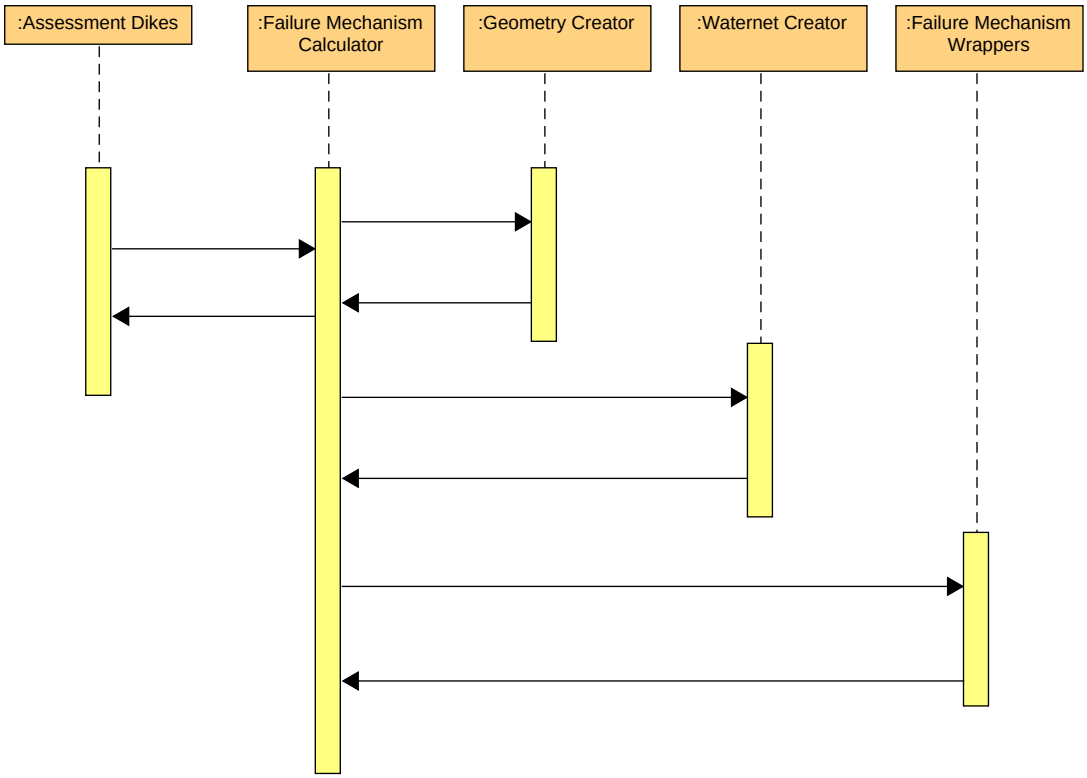

Figure 2.4: *DAM Engine Assessment sequence diagram.*

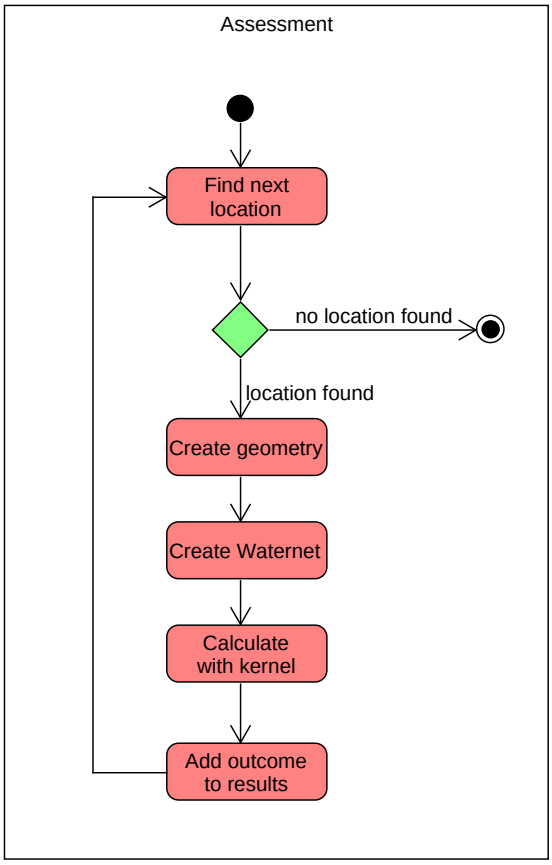

Figure 2.5: *DAM Engine Assessment activity diagram.*

2.4.2 Dikes assessment Regional

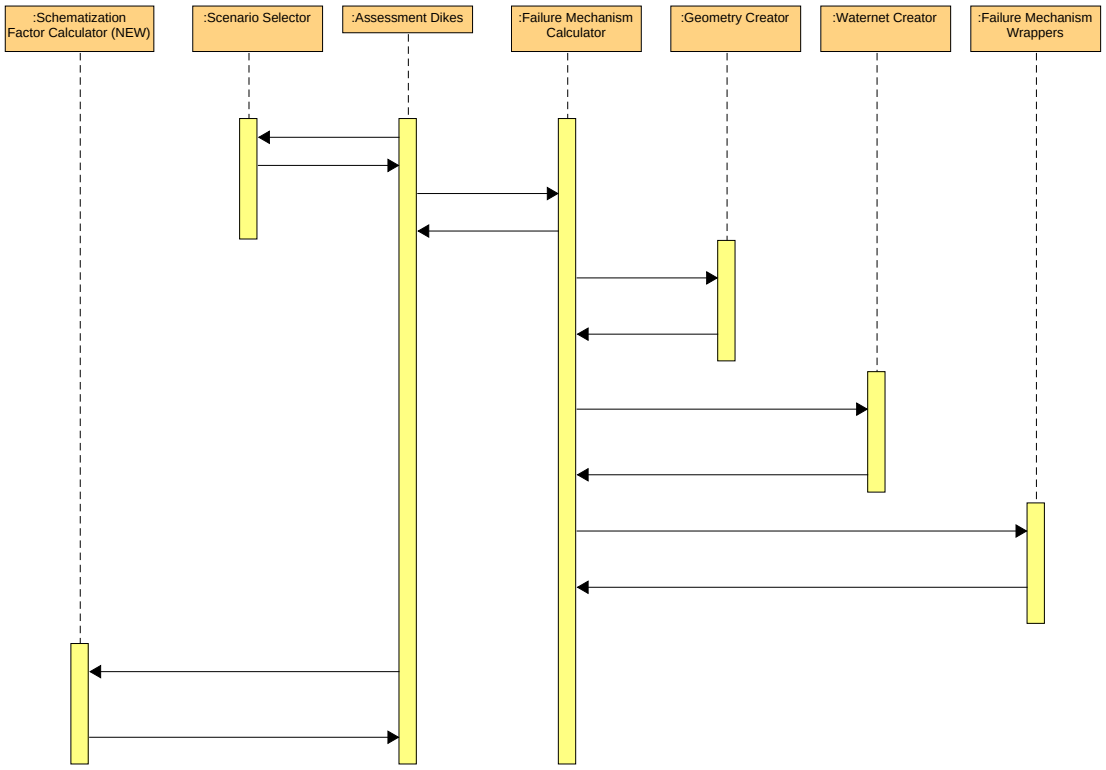

Figure 2.6: *DAM Engine Regional assessment sequence diagram.*

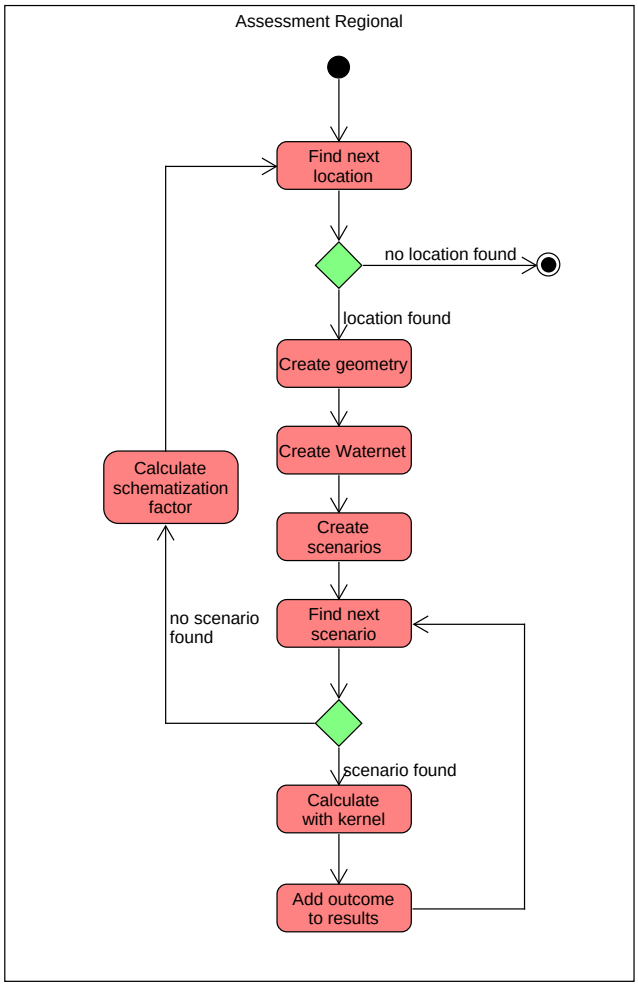

Figure 2.7: *DAM Engine Regional assessment activity diagram.*

2.4.3 Dikes operational

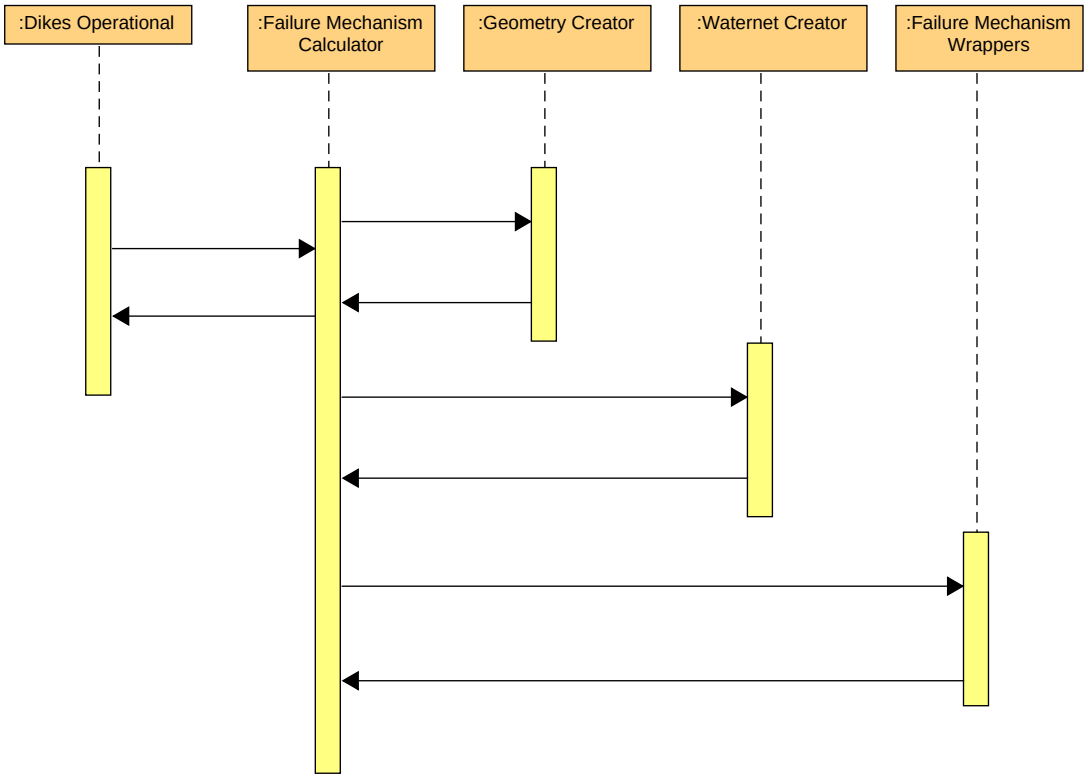

Figure 2.8: *DAM Engine Operational sequence diagram.*

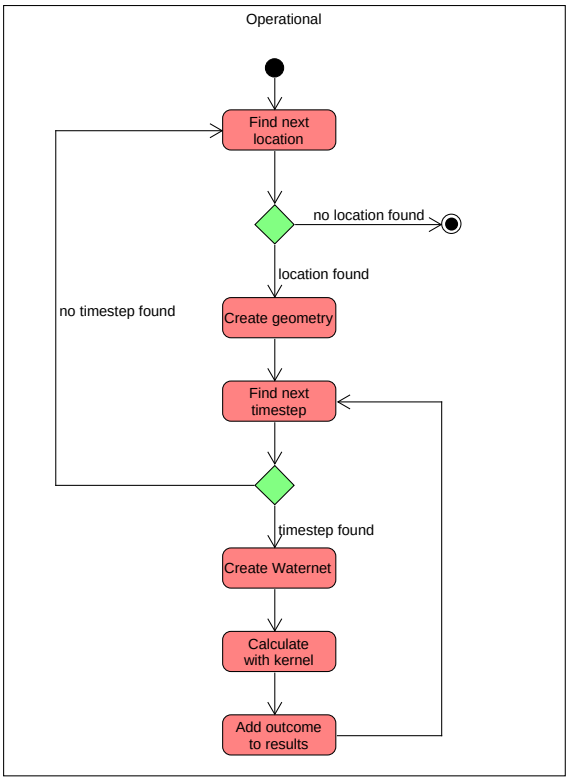

Figure 2.9: *DAM Engine Operational activity diagram.*

2.4.4 Dikes design

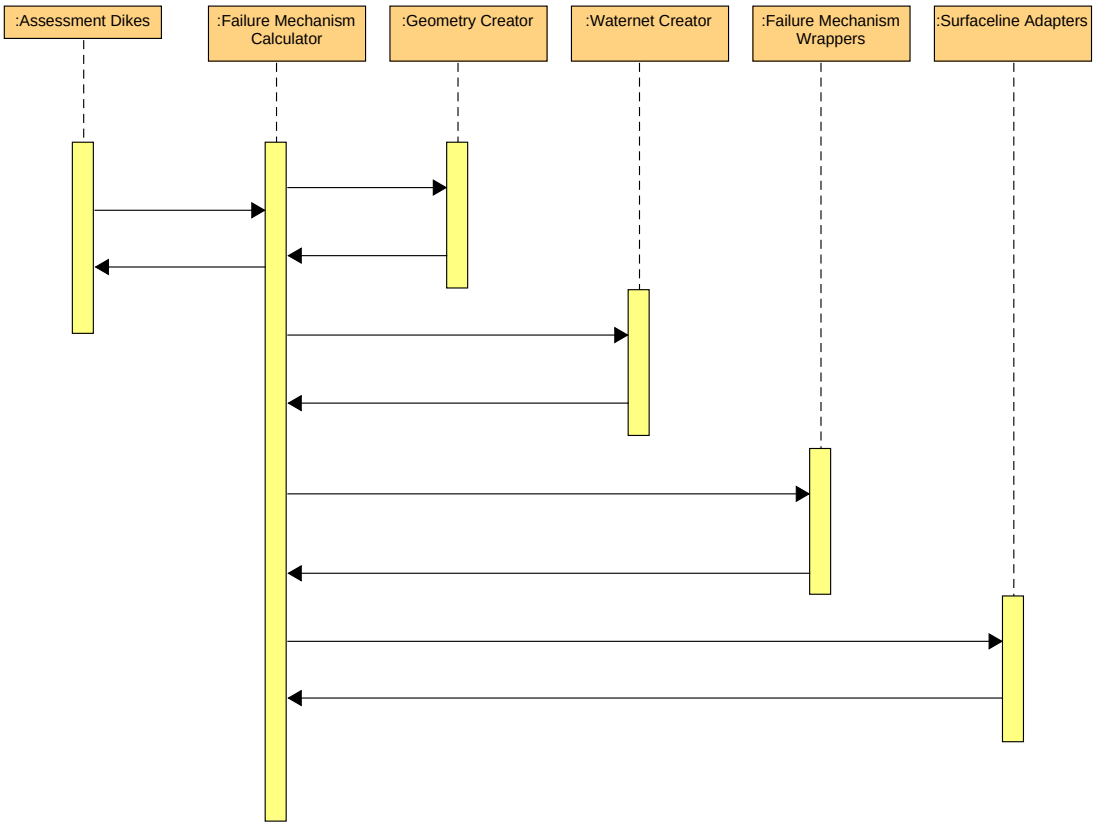

Figure 2.10: *DAM Engine Design sequence diagram.*

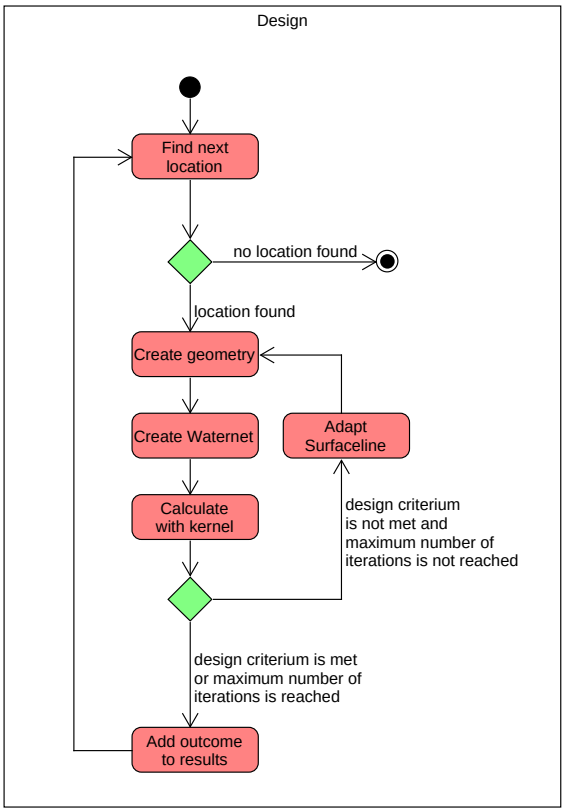

Figure 2.11: *DAM Engine Design activity diagram.*

2.4.5 Dikes NWO calculation

This will not be part of the first implementation of DAM Engine and therefor this paragraph has not yet been written.

3 Architectural Choices

3.1 Architecture guidelines

Within Deltares, DSC, a document is being written about Archtitecture Guidelines (Kleijn *et al.*, 2017). Although it is still a work in progress DAM Engine should adhere to those guidelines. More specific guidelines are added in the following sections of this chapter.

3.2 Design principles

- No circular references between objects. When it is really unavoidable, then do it through a generic interface (e.g. IParentObject)
- The calculation will support parallellization. So do not use global variables and avoid using statics.
- Failure mechanisms will be connected through wrapper classes, which will share a common IFailureMechanism interface
- Surfaceline designer classes will share a common ISurfacelineDesigner interface
- The DAM Engine must provide progress information of the calculation, so clients of the DAM Engine can show a progressbar
- The DAM Engine must provide the possiblity to abort a calculation within a reasonable timespan.
- There should be no User Interface elements shown anytime during the calculation.

3.3 Programming environment

The DAM Engine will be developed in C# with the .NET 4.5 framework. The development environment will be Visual Studio 2015.

3.4 Error handling

Errors within the DAM Engine are handled through the standard exception handling of the .NET framework. Error messages must contain as much information as possible, so a user can trace back the error to the input data.
Errorhandling with a failuremechanism kernel is done through the mechanism that is supplied by the API of the specific kernel.
Errorhandling with DAM Client is done by passing the error messages as part of the output XML file.
In fact it is the same mechanism that is used for exchanging the regular data (input and output), as shown in Figure 2.2.

The DAM Engine should be able to issue the error messages in different languages. In the first implementation only the 2 following languages will be supported:

- Dutch (NL)
- English (US)

For translations, the standard Windows mechanism with language resource dll's will be used. Note: the current implementation of DAM uses another mechanism for translations, that will not be applied here, because it is dependent on the DSL (Delta Shell Light) library, which will not be used for the DAM Engine.

3.5 Libraries and components

DAM Engine uses other libraries and components.

For now we foresee only the use of the following libraries:

- Failure mechanisms.
- Math.NET.

Other libraries may be used under the condition that they are open source and free components, that are free to redistribute.
All libraries should be listed in a manifest accompanying the release of DAM Engine. The list should also specify under which license each specific library is distributed.

Note: the current implementation of DAM uses the DSL (Delta Shell Light) library. This library will explicitly not be used for the DAM Engine, because this library is being made obsolete.

3.5.1 Failure mechanisms

The failure mechanisms are treated as external libraries. Some failure mechanisms were part of the source of the original DAM program. With the new architecture of DAM Engine this will longer be the case. These failure mechanisms will be placed in a DAM failure mechanisms library, that is not part of DAM Engine anymore. The following failure mechanisms are currently supported by the original DAM program:

- Piping Bligh (not opensource)
- Piping Sellmeijer VNK (not opensource)
- Piping Sellmeijer 2 forces (not opensource)
- D-Geo Stability inward (not opensource, but commerical)
- D-Geo Stability outward (not opensource, but commerical)

After the original failure mechanisms have been implemented, the new WBI failure mechanisms will be added:

- WTI Piping
- WTI Macrostability inward

3.5.2 Math.Net

Math.Net is a library that is distributed under the MIT/X11 license. Therefore it meets the conditions about open source and free redistribution.

4 Data Model

This chapter contains diagrams describing the main data objects of the DAM Engine and their relation to each other. In chapter 5 a short description of these data objects is given.

4.1 Main Data Model

The main data model can be seen in see Figure 4.1 It is not fully worked out, but just a global overview. The details will be filled in when programming the DAM Engine. This is because we do not intend to write a big upfront design.

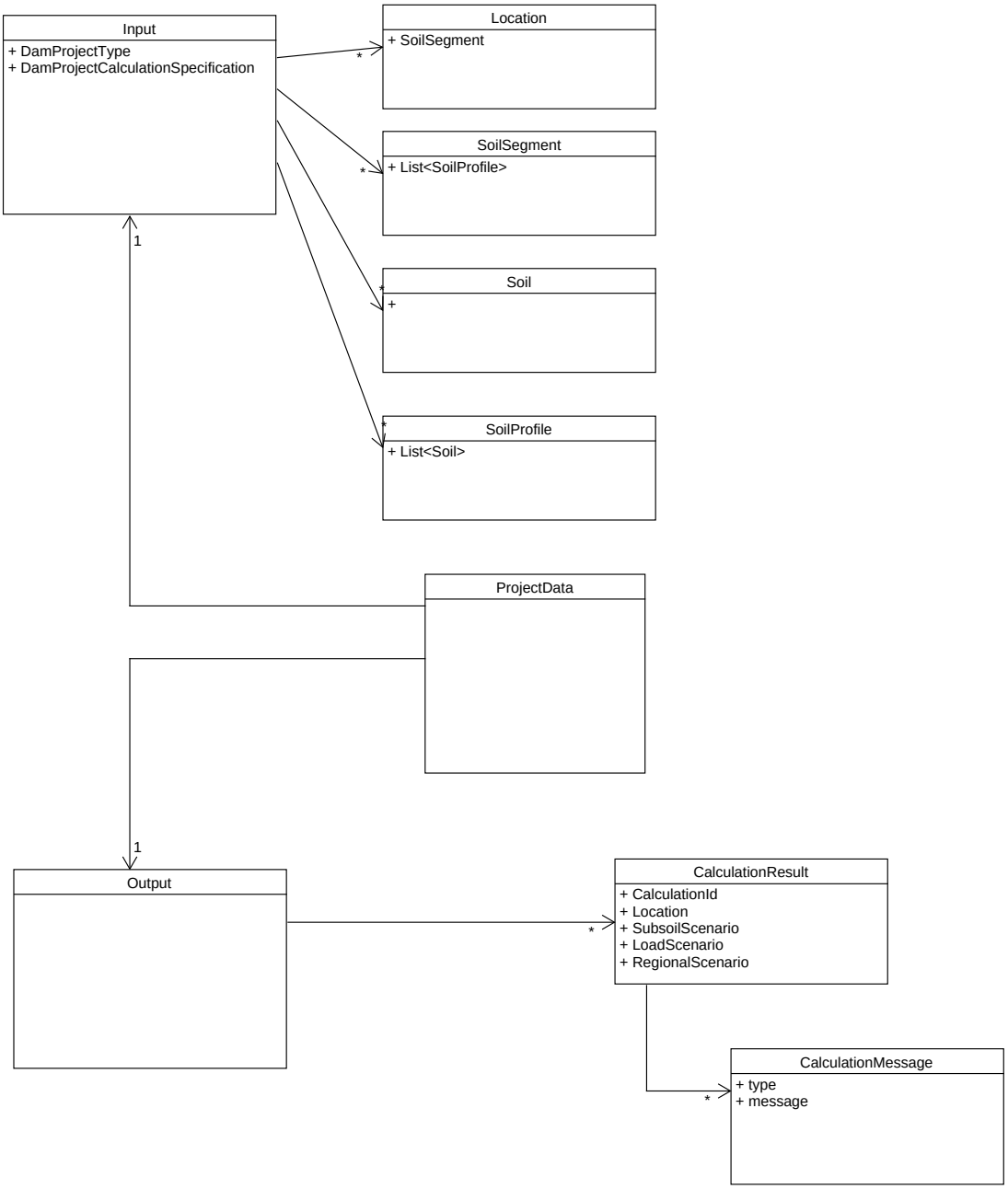

Figure 4.1: *DAM Engine main data model.*

4.2 Location

The data model of the Location class can be seen in see Figure 4.2

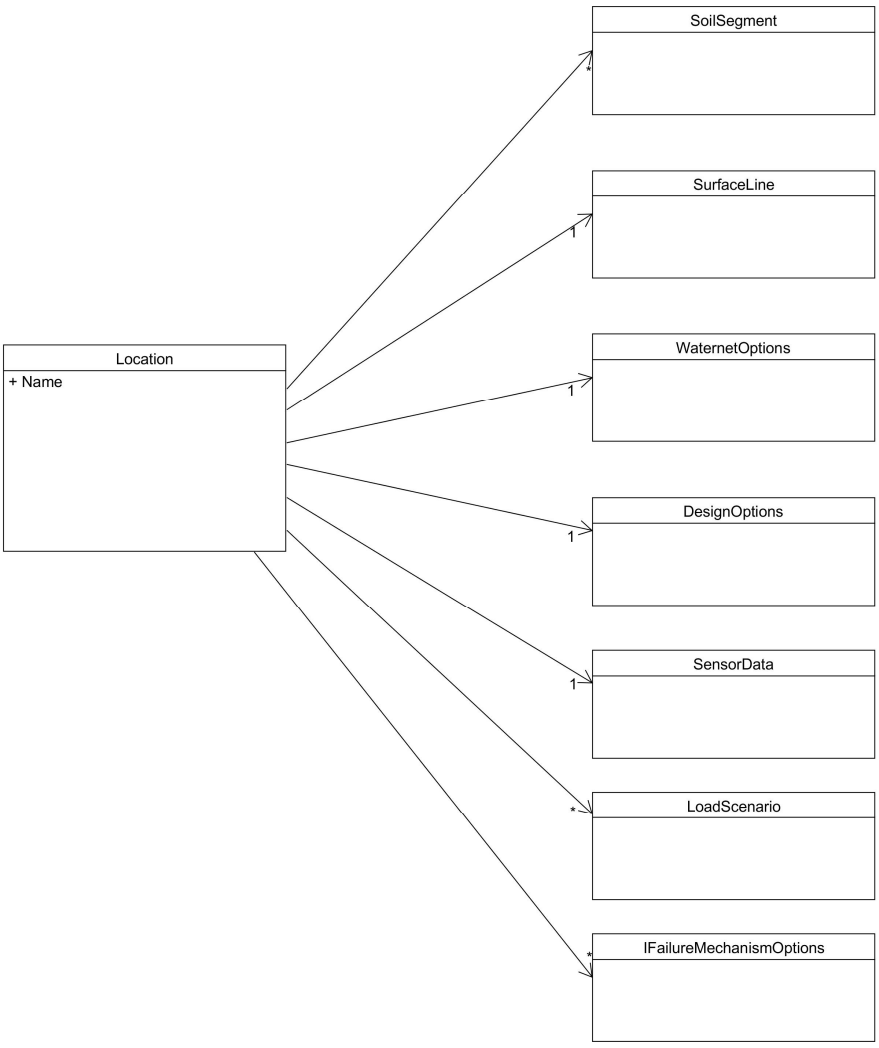

Figure 4.2: *DAM Engine Location object.*

5 Data Description

5.1 Type enumerations

5.1.1 MainMechanismType

- Macrostability inward
- Macrostability outward
- Piping

5.2 Scenarios

Scenarios are widely (a)bused within DAM. It is good to define in which context scenarios are used and how they are to be called. Simply using the word scenario is not enough. Within DAM we have 3 types of scenarios:

- Subsoil scenario
- Design scenario
- Regional scenario

5.2.1 Subsoil scenario

Used as part of the stochastic subsoil schematization. A subsoil scenario defines a possible 1D- or 2D-profile that applies to a certain location.

5.2.2 Design scenario

Used for Design calculation. In a design calculation a new surfaceline is designed for a location, based on a target failure factor (e.g. due to new requirements), or load (e.g. a higher waterlevel).

5.2.3 Regional scenario

For regional assessments there are certain scenarios that have to be evaluated, depending on the circumstances (e.g. drought, type of dike etc.). Part of the assessment is the determination which scenarios have to be evaluated and then calculating those scenarios.

5.3 Main Data Model

5.3.1 Input

DamProjectType

- Assessment
- Assessment regional
- Operational
- Design
- NWO

DamProjectCalculationSpecification

This class specifies which failuremechanism is to be calculated and it also contains the specific options for the selelected mechanism (e.g. which calculation model)

Locations

This is a collection of locations, with each location containing the location specific data.

Soil Segments

This is a collection of soil segments, with each segment containing the subsoil data for a specific failure mechanism.

Soils

This is a collection of soils, with each soil containing the soil parameters needed for the calculation of all failure mechanisms.

5.3.2 Output

CalculationResults

A calculation result holds the result for a specific location, a specific failure mechanism, and a specific subsoil scenario of a specific segment defined in the location data.

CalculationMessages

These are all the message that are generated by the calculation. A message must contain as much information as possible to trace back the information tho the input data (e.g. a specific location, a specific failure mechanism, and a specific subsoil scenario of a specific segment defined in the location data).

5.4 Location

SoilSegment

A soil segment contains the subsoil data for a specific failure mechanism.

SurfaceLine

A surfaceline describes the dike profile in a specific location. In the Design calculation it can also be the new dike profile, which can meet design criteria in a specific design scenario.

WaternetOptions

The options that support the creation of a waternet in a specific location.

DesignOptions

The options that will be used in the Design calculation (e.g. how to design a shoulder when needed).

SensorData

The sensor data can be used to define a waternet based on live sensor data. This sensor data holds information about ID and location of the sensor. The actual sensor readings are defined as timeseries readings for each sensor in each location.

DesignScenario

Used for Design calculation. A design scenario contains the following items:

- Riverlevel low
- Riverlevel high
- Dike table height
- Required safety factor for each specified failure mechanism
- Uplift criterium for each specified failure mechanism
- Waternet options for each specified failure mechanism

IFailureMechanismOptions

Specific options for each location for each failure mechanism.

6 Module Description

6.1 DAM Engine main modules

6.1.1 Assessment Dikes

This module performs an assessment for general dikes (e.g. primary dikes).

Primary assessment calculation

This is the main submodule of the primary assessement. This submodule contains the main loop of the calculation.

6.1.2 Assessment Regional Dikes

This module performs an assessment for regional dikes.

Regional assessment calculation

This is the main submodule of the regional assessement. This submodule contains the main loop of the calculation.

Regional scenario selector

This submodule generates all the scenarios that have to be evaluated for a specific location. The scenarios are selected based on the local conditions.

Regional schematization factor calculator

This submodule calculates the schematization factor in a location based on all results of all scenarios in a location.

6.1.3 Design Dikes

This module performs an design calculation for all types of dikes.

Primary design calculation

This is the main submodule of the primary design calculation. This submodule contains the main loop of the calculation.

6.1.4 Operation module

This module performs a time series based calculation for all types of dikes.

Time series based calculation

This is the main submodule of the time series based calculation. This submodule contains the main loop of the calculation.

6.1.5 NWO Calculation

This module performs an NWO (Niet Waterkerende Objecten) calculation for primary dikes.

Primary NWO calculation

This is the main submodule of the NWO calculation. This submodule contains the main loop of the calculation.

6.2 DAM Engine supporting modules

6.2.1 Failure mechanism wrapper interface

For each failuremechanism kernel a specific wrapper will be written. This wrapper must implement a specific interface, so the DAM Engine can support the use of the failuremechanism kernel. The interface that must be implemented is IFailureMechanism.
Example: Lets say that for the failure mechanism piping we have 3 kernels: Bligh, Sellmeijer and VNK. Then for each of these kernels a calculation wrapper has to be written.
Another example: D-Geo Stability kernel has the ability to calculate the failure mechanism macrostability inwards and the failure mechanism macrostability outwards. In this case 2 wrappers (one for each failure mechanism) are needed for this single kernel.
The next methods are defined in the IFailureMechanism interface

- Prepare()
- Validate()
- Execute()
- Design()
- PostProcess()
- RegisterProgressFeedback()
- RegisterAbortCheck()

6.2.1.1 Prepare

The purpose of this method is to fill a dataobject that implements the IKernelDataInput interface. This dataobject will be needed for the other methods in this interface. This method has one parameter:

- Input

It returns IKernelDataInput. This method fills the IKernelDataInput object from the main input data object (Input). In IKernelDataInput the data is filled that is needed by the specific failuremechanism kernel data. Each failuremechanism kernel wrapper will have its own implementation of IKernelDataInput.

6.2.1.2 Validate

The purpose of this method is to validate the data that will be used as input for the failure mechanism. This method has 2 parameters:

- IKernelDataInput (kernel input data)
- ValidationMessageList (a list of messages produced by the validation)

It returns an integer: 0: no errors. A calculation is possible. It is possible that there are warning messages. >0: number of error messages that prevent a calculation.

6.2.1.3 Execute

This method performs the actual calculation of the failure mechanism, This method has 2 parameters:

- IKernelDataInput (kernel input data)
- ErrorMessageList (a list of error messages produced by the calculation)

It returns IKernelDataOutput. IKernelDataOutput contains the calculation results of the failuremechanism kernel data. Each failuremechanism kernel wrapper will have its own implementation of IKernelDataOutput.

6.2.1.4 Design

This method implements a design calculation. Based on certain design parameters (e.g. target failure factor, new load parameters, design strategies, etc.) a new design is made for the input data (e.g. a new surfaceline). This method has 3 parameters:

- IKernelDataInput (kernel input data)
- IKernelDesignDataInput (definition of the design parameters)
- ErrorMessageList (a list of error messages produced by the design calculation)

It returns a IKernelDesignDataOutput. IKernelDesignDataOutput contains the adapted input data (a.g. a new designed surfaceline) and other design results (e.g. number of iterations needed, success or failure etc.).
Based on the given criteria a new design is determined, which will meet the required criteria. If such a design is not possible, that will be reported back.

6.2.1.5 PostProcess

This method has 2 parameters

- IKernelDataOutput (kernel output data)
- Output (the DAM Engine output data)

This method fills the DAM Engine Output object with the results of the failuremechanism kernel (IKernelDataOutput).

6.2.1.6 RegisterProgressFeedback

This method registers a callback function into the failuremechanism kernel wrapper that can report back progress status from the failuremechanism kernel wrapper to the calling application. The calling application provides the callback function that should be called.

6.2.1.7 RegisterAbortCheck

This method registers a callback function into the failuremechanism kernel wrapper. The calling application provides the callback function that should be called. If the function reports back that an abort was requested, the failuremechanism kernel should abort the calculation and return to the calling application with an appropriate error message.

6.2.2 Failure mechanism wrapper implementations

For now the next three implementations of failure mechanism wrappers are foreseen. In the future more can be added. Note also that for a specific failure mechanism multiple implementations can be created. E.g. Piping:

- piping Bligh
- piping Sellmeijer 2 forces
- piping Sellmeijer 4 forces
- piping VNK model

Macrostability inwards

Calculation wrapper for Macrostability inward. Note that (as already mentioned above) for each specific kernel implementation for a failure mechanism, a separate wrapper has to be build (e.g. Slope/W and D-Geo Stability)

Macrostability outwards

Calculation wrapper for Macrostability outward.

Piping

Calculation wrapper for Piping.

6.2.3 Surfaceline designers

A collection of surfaceline designers to support the design calculation. Each designer should adhere to the ISurfaceLineDesigner interface.

Surfaceline Designer Height

Adapts the surfaceline by adding extra height to the dike crest.

Surfaceline Designer Slope

Adapts the surfaceline by changing the slope of the dike on the inside.

Surfaceline Designer Shoulder

Adapts the surfaceline by adding a shoulder or enlarging the shoulder on the inside of the dike.

Surfaceline Designer NWO

Adapts the surfaceline by adding a NWO on a specific place in the surfaceline.

6.2.3.1 Calculation Runner

Failure mechanism Calculation Runner

This submodule calculates a specific failure mechanism by calling the IFailureMechanism interface of the wrapper implementation.

Design Calculation Runner

This submodule performs a design calculation for a specific failure mechanism by calling the IFailureMechanism interface and several surfaceline designers through their ISurfacelineDesigner interface.

Operational Calculation Runner

This submodule can perform a calculation based on sensor readings (as time-series). The load on the systems (the waternet) will be based on those sensor readings. This can be used in operational systems like DamLive.

Probabilistic Calculation Runner

This submodule performs a probabilistic calculation for a specific location and failure mechanism. The outcome is a failure probability for that location and failure mechanism.

6.2.4 General submodules

Geometry creator

This submodule combines a surfaceline with a subsoil scenario. The output is a geometry that can be used by the failure mechanisms to perform a calculation.

Waternet creator

A waternet describes the waterpressures in the dike embankment. The waterpressures are a result of the load on the system (outer waterlevel and polderlevel). This submodule determines the waternet that will be used by the failure mechanism kernels. At first only the current DAM implementation will be used as a waternet creator. Later on new implementations can be made and applied. E.g. specific for each failure mechanism, or an implementation based on a numerical model like DgFlow.

6.2.4.1 Scripting engine

To enable advanced users to experiment with how the DAM Engine works a Python scripting engine is implemented as a submodule. The scripting engine has acces to the data model and the submodules through well defined interfaces.

7 Programming Interface

This is the definition of the programming interface. The only way to communicate to the DAM Engine is through this interface. In the assembly Deltares.DamEngine.Interface.dll a class hass been defined: `Interface`, which provides the properties and methods which can be used to interact with the DAM Engine.

7.1 Initialization

```
/// <summary>
/// Initializes a new instance of the <see cref="Interface"/> class.
/// </summary>
/// <param name="modelInput">Xml string containing the model input.</param>
public Interface(string modelInput)
```

The class has to be instantiated with an (XML) string which adheres to the XSD definition of the inputfile for the DAM Engine (See Appendix A).

7.2 Validation

```
/// <summary>
/// Validates the model.
/// </summary>
/// <returns>Validation messages in an XML string</returns>
public string Validate()
```

This will validate the model and returns the messages in an XML string which adheres to the XSD definition of a message list (See Appendix B).

7.3 Calculation

```
/// <summary>
/// Performs the calculation.
/// </summary>
/// <returns>The output of the calculation in an XML string</returns>
public string Run()
```

This will perform the calculation of the model and returns an XML string which adheres to the XSD definition of the output of the DAM Engine (See Appendix C).

7.4 Interaction

The DAM Engine interacts with the calling application through delegates. The following delegates are used by the DAM Engine:

```
/// <summary>
/// Sends the current progress status
/// </summary>
/// <param name="progress">The progress; this is a number between 0 and 1.</param>
public delegate void ProgressDelegate(double progress);

/// <summary>
/// Sends log message
/// </summary>
/// <param name="logMessage">The log message.</param>
public delegate void SendMessageDelegate(LogMessage logMessage);

/// <summary>
/// Check if a user abort is requested
/// </summary>
/// <returns>true if user requested an abort; else false</returns>
public delegate bool UserAbortDelegate();
```

These delegates can be assigned to the properties of Interface:

```
public ProgressDelegate ProgressDelegate
public SendMessageDelegate SendMessageDelegate
public UserAbortDelegate UserAbortDelegate
```

8 Literature

Doxygen, 2017. *DAM Engine - Technical documentation, Generated by Doxygen 1.8.10*. Tech. rep., Deltares.

Kleijn, E., A. Grijze, H. Elzinga, S. Hummel and T. The, 2017. *Architecture Guidelines*. Tech. rep., Deltares.

The, T., 2017a. *DAM Architecture Overall*. Tech. Rep. 1210702-000-GEO-0005, version 0.1, jan. 2017, concept, Deltares.

The, T., 2017b. *DAM Engine - Technical Design*. Tech. Rep. 1210702-000-GEO-0004, version 0.2, mar. 2017, concept, Deltares.

Trompille, V., 2017a. *DAM Engine - Test Plan*. Tech. Rep. 1210702-000-GEO-0006, version 0.1, jan. 2017, concept, Deltares.

Trompille, V., 2017b. *DAM Engine - Test Report*. Tech. Rep. 1210702-000-GEO-0007, version 0.1, jan. 2017, concept, Deltares.

Zwan, I. v., 2017. *DAM Engine - Functional Design*. Tech. Rep. 1210702-000-GEO-0003, version 0.1, jan. 2017, concept, Deltares.

A DamInput

These are the XSD's that apply to the input XML of the DAM Engine.

A.1 DamInput.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid XML Studio 1.0.8.0 (http://www.liquid-technologies.com) -->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation=".\DamLocation.xsd" />
  <xs:complexType name="Input">
    <xs:sequence>
      <xs:element name="Locations">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded" name="Location" type="Location" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="DamProjectType" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Assessment" />
          <xs:enumeration value="AssessmentRegional" />
          <xs:enumeration value="Operational" />
          <xs:enumeration value="Design" />
          <xs:enumeration value="NWO" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="Input" />
</xs:schema>
```

A.2 DamLocation.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid XML Studio 1.0.8.0 (http://www.liquid-technologies.com) -->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Location">
    <xs:sequence>
      <xs:element name="AssesmentRegionalOptions">
        <xs:complexType>
          <xs:attribute default="Clay" name="DikeMaterialType">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="Sand" />
                <xs:enumeration value="Peat" />
                <xs:enumeration value="Loam" />
                <xs:enumeration value="Clay" />
                <xs:enumeration value="Gravel" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="DredgingDepth" type="xs:double" />
          <xs:attribute default="0.95" name="DetrimentFactor" type="xs:double" />
          <xs:attribute name="DikeTableHeight" type="xs:double" />
        </xs:complexType>
      </xs:element>
      <xs:element name="AssesmentOptions" />
      <xs:element name="OperationalOptions" />
      <xs:element name="DesignOptions" />
```

```
<xs:element name="NWOOptions" />
<xs:element name="WaternetOptions">
  <xs:complexType>
    <xs:attribute name="PhreaticLineCreationMethod" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ExpertKnowledgeRRD" />
          <xs:enumeration value="ExpertKnowledgeLinearInDike" />
          <xs:enumeration value="GaugesWithFallbackToExpertKnowledgeRRD" />
          <xs:enumeration value="Sensors" />
          <xs:enumeration value="None" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="PolderLevel" type="xs:double" use="required" />
    <xs:attribute name="HeadPl2" type="xs:double" use="optional" />
    <xs:attribute name="HeadPl3" type="xs:double" use="optional" />
    <xs:attribute name="HeadPl4" type="xs:double" use="optional" />
    <xs:attribute name="DampingFactorPL3" type="xs:double" use="required" />
    <xs:attribute name="DampingFactorPL4" type="xs:double" use="required" />
    <xs:attribute name="PenetrationLength" type="xs:double" use="required" />
    <xs:attribute name="SlopeDampingFactor" type="xs:double" use="required" />
    <xs:attribute name="Pl1BelowCrestRiverside" type="xs:double" use="required" />
    <xs:attribute name="Pl1BelowCrestPolderside" type="xs:double" use="required" />
    <xs:attribute name="Pl1BelowShoulderCrestPolderside" type="xs:double" use="required" />
    <xs:attribute name="Pl1BelowToeDikePolderside" type="xs:double" use="required" />
    <xs:attribute name="Pl1BelowCrestMiddle" type="xs:double" use="optional" />
    <xs:attribute name="Pl1FactorBelowShoulderCrest" type="xs:double" use="optional" />
    <xs:attribute name="DryPl1BelowCrestMiddle" type="xs:double" use="optional" />
    <xs:attribute name="DryPl1FactorBelowShoulderCrest" type="xs:double" use="optional" />
    <xs:attribute name="IntrusionVerticalWaterPressure" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Standard" />
          <xs:enumeration value="Linear" />
          <xs:enumeration value="FullHydroStatic" />
          <xs:enumeration value="HydroStatic" />
          <xs:enumeration value="SemiTimeDependent" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="General">
  <xs:complexType>
    <xs:attribute default="Primary" name="DamType">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Primary" />
          <xs:enumeration value="Regional" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Description" type="xs:string" />
    <xs:attribute name="DikeEmbankmentMaterial" type="xs:string" />
    <xs:attribute default="1.0" name="ForbiddenZoneFactor" type="xs:double" />
    <xs:attribute name="IsUseOriginalPLLineAssignments" type="xs:boolean" />
    <xs:attribute name="HeadPL2" type="xs:double" use="optional" />
    <xs:attribute name="HeadPL3" type="xs:double" use="optional" />
    <xs:attribute name="headPL4" type="xs:double" use="optional" />
  </xs:complexType>
</xs:element>
<xs:element name="PipingProbabilisticParameters">
  <xs:complexType>
    <xs:attribute default="Deterministic" name="LayerHeightDistribution">
      <xs:simpleType>
```

```
            <xs:restriction base="xs:string">
              <xs:enumeration value="Deterministic" />
              <xs:enumeration value="Uniform" />
              <xs:enumeration value="Triangular" />
              <xs:enumeration value="Normal" />
              <xs:enumeration value="LogNormal" />
              <xs:enumeration value="Exponential" />
              <xs:enumeration value="Gamma" />
              <xs:enumeration value="Beta" />
              <xs:enumeration value="Frechet" />
              <xs:enumeration value="Weibull" />
              <xs:enumeration value="Gumbel" />
              <xs:enumeration value="Rayleigh" />
              <xs:enumeration value="Pareto" />
              <xs:enumeration value="TruncatedNormal" />
              <xs:enumeration value="Table" />
              <xs:enumeration value="StudentT" />
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="LayerHeightDeviation" type="xs:double" />
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

B MessageList

C DamOutput

C.1 DamOutput.xsd