

# Implementierungsbericht

ROSE (Road System Editor)  
Edit Highway Roads Easily



Planung: Jannes Wagner  
Entwurf: Yannik Sproll  
Implementierung: Philipp Seidel  
Qualitätssicherung: Cristian Gorun  
Abnahme: Max Schweikart

Betreuung: Erik Burger, Jelle Kübler und Marvin Baumann

Institut für Verkehrswesen  
Fakultät für Bauingenieur-, Geo- und Umweltwissenschaften  
Karlsruher Institut für Technologie

November 2021

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>5</b>
<b>2 Implementierungsplan</b>	<b>6</b>
<b>3 Veränderungen am Entwurf</b>	<b>8</b>
3.1 Korrektur fehlerhafter Signaturen . . . . .	8
3.1.1 edu.kit.rose.infrastructure.language . . . . .	8
3.1.2 edu.kit.rose.view.panel.segment . . . . .	8
3.1.3 edu.kit.rose.view.panel.segmentbox . . . . .	8
3.1.4 edu.kit.rose.controller.commons . . . . .	8
3.1.5 edu.kit.rose.controller.hierarchy . . . . .	9
3.1.6 edu.kit.rose.controller.plausibility . . . . .	9
3.1.7 edu.kit.rose.controller.project . . . . .	9
3.1.8 edu.kit.rose.controller.roadsystem . . . . .	9
3.1.9 edu.kit.rose.model . . . . .	10
3.1.10 edu.kit.rose.model.plausibility.criteria . . . . .	10
3.1.11 edu.kit.rose.model.roadsystem . . . . .	10
3.2 Einfügen fehlender Signaturen . . . . .	10
3.2.1 edu.kit.rose.view.common . . . . .	11
3.2.2 edu.kit.rose.view.window . . . . .	11
3.2.3 edu.kit.rose.controller.attribute . . . . .	11
3.2.4 edu.kit.rose.controller.command . . . . .	11
3.2.5 edu.kit.rose.controller.hierarchy . . . . .	11
3.2.6 edu.kit.rose.controller.roadsystem . . . . .	11
3.2.7 edu.kit.rose.controller.plausibility . . . . .	12
3.2.8 edu.kit.rose.controller.project . . . . .	12
3.2.9 edu.kit.rose.model.roadsystem . . . . .	12
3.3 Verwendung von JavaFx . . . . .	12
3.3.1 edu.kit.rose.infrastructure . . . . .	12
3.3.2 edu.kit.rose.view.common . . . . .	12
3.3.3 edu.kit.rose.view.panel.criterion . . . . .	13
3.3.4 edu.kit.rose.view.panel.hierarchy . . . . .	13
3.3.5 edu.kit.rose.view.panel.segmentbox . . . . .	13
3.4 Integration von Dependency Injection . . . . .	13
3.4.1 edu.kit.rose.view.common . . . . .	13
3.4.2 edu.kit.rose.view.window . . . . .	13
3.5 Verbesserung der Codequalität . . . . .	14
3.5.1 edu.kit.rose.infrastructure . . . . .	14
3.5.2 edu.kit.rose.view.common . . . . .	14
3.5.3 edu.kit.rose.panel.criterion . . . . .	14
3.5.4 edu.kit.rose.panel.segment . . . . .	14

3.5.5	edu.kit.rose.panel.violation . . . . .	14
3.5.6	edu.kit.rose.controller.attribute . . . . .	15
3.5.7	edu.kit.rose.controller.command . . . . .	15
3.5.8	edu.kit.rose.controller.commons . . . . .	15
3.5.9	edu.kit.rose.controller.plausibility . . . . .	15
3.5.10	edu.kit.rose.view.panel.roadsystem . . . . .	15
3.5.11	edu.kit.rose.model . . . . .	15
3.5.12	edu.kit.rose.model.roadsystem.elements . . . . .	16
3.6	Änderungen durch Implementierungskontext . . . . .	16
3.6.1	edu.kit.rose.infrastructure . . . . .	16
3.6.2	edu.kit.rose.view.commons . . . . .	16
3.6.3	edu.kit.rose.view.panel.hierarchy . . . . .	16
3.6.4	edu.kit.rose.view.panel.segmentbox . . . . .	16
3.6.5	edu.kit.rose.view.panel.violation . . . . .	17
3.6.6	edu.kit.rose.controller.navigation . . . . .	17
3.6.7	edu.kit.rose.controller.project . . . . .	17
3.6.8	edu.kit.rose.model . . . . .	17
3.6.9	edu.kit.rose.model.roadsystem . . . . .	18
3.6.10	edu.kit.rose.model.roadsystem.attributes . . . . .	18
3.6.11	edu.kit.rose.model.roadsystem.elements . . . . .	18
<b>4</b>	<b>Abdeckung der Muss/Wunschkriterien und Anforderungen</b>	<b>19</b>
4.1	Umgesetzte Musskriterien . . . . .	19
4.2	Umgesetzte Wunschkriterien . . . . .	19
4.3	Abgedeckte Funktionale Anforderungen . . . . .	20
4.3.1	Das Programm kann Straßennetze graphisch darstellen. . . . .	20
4.3.2	Der Nutzer kann das Straßennetz aus §4.3.2 bearbeiten. . . . .	20
4.3.3	Das Programm prüft das Straßennetz auf Plausibilität. . . . .	21
4.3.4	Der Nutzer kann das Projekt exportieren. . . . .	22
4.4	Der Nutzer kann Projekte speichern und laden. . . . .	22
4.4.1	Weitere Anforderungen . . . . .	22
4.5	Abgedeckte Nichtfunktionale Anforderungen . . . . .	23
4.6	Nicht umgesetzte Wunschkriterien . . . . .	23
4.7	Nicht abgedeckte Funktionale Anforderungen . . . . .	24
4.8	Nicht abgedeckte Nichtfunktionale Anforderungen . . . . .	25
<b>5</b>	<b>Veränderungen am Implementierungsplan</b>	<b>26</b>
<b>6</b>	<b>Arbeitsabläufe</b>	<b>28</b>
6.1	Workflow . . . . .	28
6.2	Durchsichten . . . . .	28
6.3	Entwicklungsumgebung . . . . .	29
6.4	Modultests . . . . .	30

<b>7 Statistiken</b>	<b>32</b>
<b>Glossar</b>	<b>34</b>

# **1 Einleitung**

Dieses Dokument enthält den Abschlussbericht zur Implementierungsphase des PSE-Projekts “ROSE”. Die diesem Dokument zugrunde liegende Implementierung setzt die im ROSE-Pflichtenheft spezifizierten Anforderungen um. Diese Implementierung orientiert sich bei der Umsetzung am im ROSE-Entwurfsdokument festgelegten Entwurf.

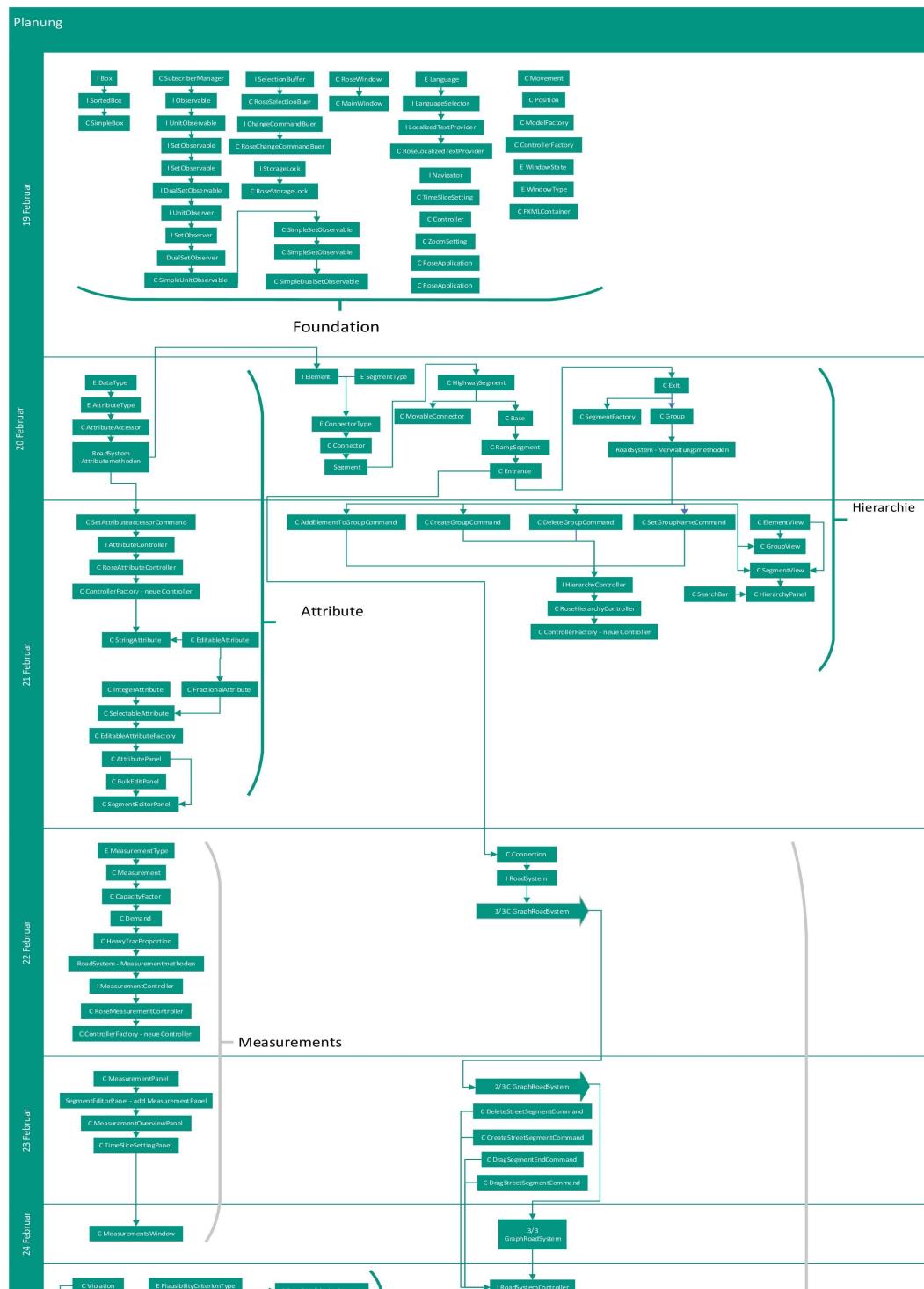
Der in diesem Dokument enthaltene Implementierungsplan zeigt die anfänglich geplante, zeitliche Einteilung der Implementierungen der einzelnen, vom ROSE-Entwurfsdokument vorgegebenen Klassen. Zusätzlich dazu wird die tatsächlich benötigte Implementierungszeit dargelegt. Die über Implementierungszeit und Codezeilenanzahl erhobenen Statistiken werden hier ebenfalls aufgeführt.

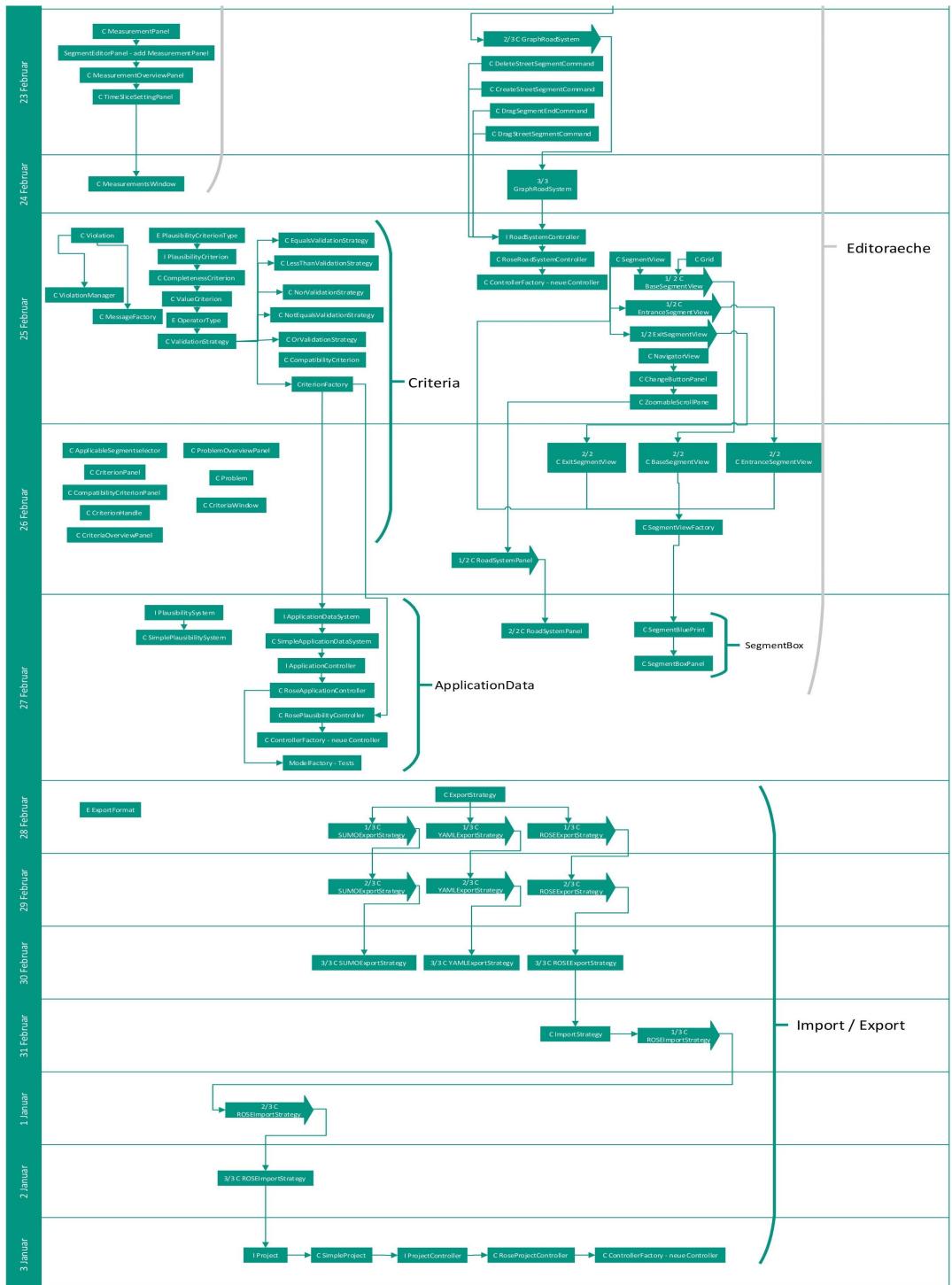
Des Weiteren enthält dieses Dokument eine Übersicht über Änderungen und Ergänzungen des im ROSE-Entwurfsdokument festgelegten Entwurfs.

Auf die im Pflichtenheft angegebenen Kriterien und Anforderungen, sowie deren Umsetzung wird in diesem Dokument auch eingegangen. Betrachtet wird dabei, welche Kriterien und Anforderungen umgesetzt bzw. nicht umgesetzt wurden. Gründe für das Nichtumsetzen eines Kriteriums bzw. einer Anforderung werden ebenfalls erörtert.

Außerdem werden die bei der Implementierung verwendeten Praktiken und Abläufe, sowie die Resultate der Testüberdeckung angegeben.

## 2 Implementierungsplan





## 3 Veränderungen am Entwurf

In diesem Abschnitt werden Änderungen, die während der Implementierungsphase am Entwurf vorgenommen wurden, festgehalten. Diese Änderungen werden hier gruppiert nach ihrem Grund aufgeführt. Für eine Gruppe von Änderungen wird immer zuerst eine Begründung festgehalten. Dann werden alle deshalb geänderten Stellen aufgelistet.

### 3.1 Korrektur fehlterhafter Signaturen

In diesem Abschnitt werden die Korrekturen an den fehlerhaften Signaturen aus dem ROSE-Entwurfsdokument aufgeführt.

#### 3.1.1 edu.kit.rose.infrastructure.language

- Die Methoden `LocalizedTextProvider.subscribeToOnLangaugeChanged` und `LocalizedTextProvider.unsubscribeFromOnLanguageChanged` haben einen Parameter vom Typ `Consumer<Language>`.

#### 3.1.2 edu.kit.rose.view.panel.segment

- Der Schutzmodifikator der Klasse `EditableAttributeFactory` ist `package private`
- Die Klasse `SegmentEditorPanel` erweitert das Interface `SetObserver<Element, Element>` anstatt dem Interface `UnitObserver<Element>`.

#### 3.1.3 edu.kit.rose.view.panel.segmentbox

- Die Klasse `SegmentBoxPanel` erweitert die Klasse `FxmlContainer`.

#### 3.1.4 edu.kit.rose.controller.commons

- Der Konstruktor der Klasse `Controller` hat keinen Parameter vom Typ `ChangeCommandBuffer` mehr.

- Der Parameter vom Typ `Navigator` wurde von der Methode `Controller.init` in den Konstruktor von `Controller` verschoben. In diesem Zuge wurde auch die Methode `Controller.init` entfernt.

### **3.1.5 `edu.kit.rose.controller.hierarchy`**

- Der Konstruktor der Klasse `SetGroupNameCommand` hat keinen Parameter vom Typ `Project` mehr.

### **3.1.6 `edu.kit.rose.controller.plausibility`**

- Der Konstruktor der Klasse `RosePlausibilityController` hat keinen Parameter vom Typ `ChangeCommandBuffer`.
- Der Konstruktor der Klasse `RosePlausibilityController` hat einen Parameter vom Typ `SelectionBuffer` und einen Parameter vom Typ `Navigator`.

### **3.1.7 `edu.kit.rose.controller.project`**

- Der Konstruktor der Klasse `RoseProjecController` hat keinen Parameter vom Typ `ChangeCommandBuffer`.
- Der Konstruktor der Klasse `RoseProjectController` hat einen Parameter vom Typ `Navigator`.

### **3.1.8 `edu.kit.rose.controller.roadsystem`**

- Der Typ des Parameters der Methode `RoadSystemController.setZoomLevel` wurde von `Integer` auf `Double` geändert.
- Die Methode `RoadSystemController.endDragStreetSegment` hat einen Parameter vom Typ `Connector`.
- Der Konstruktor der Klasse `RoseRoadSystemController` hat einen Parameter vom Typ `Navigator`.

- Die Parameter des Konstruktors der Klasse `DragSegmentEndCommand` wurden ersetzt durch einen Parameter vom Typ `MovableConnector` und einen Parameter vom Typ `Movement`.
- Der Konstruktor der Klasse `DragStreetSegmentsCommand` hat keinen Parameter vom Typ `Position`.

### **3.1.9 edu.kit.rose.model**

- Die Methoden `ApplicationDataSystem.importCriteriaFromFile` und `ApplicationDataSystem.exportCriteriaToFile` haben den Rückgabetyp boolean.
- Die Methoden `ExportStrategy.exportToFile` und `RoseExportStrategy.importToProject` haben den Rückgabetyp boolean.
- Die Methoden `Project.exportToFile`, `Project.save` und `Project.load` haben den Rückgabetyp boolean.

### **3.1.10 edu.kit.rose.model.plausibility.criteria**

- Die Methode `CriteriaManager.createCompatibilityCriterion` gibt ein `CompatibilityCriterion` zurück.

### **3.1.11 edu.kit.rose.model.roadsystem**

- Die Methode `RoadSystem.createSegment` hat den Rückgabetyp `Segment`
- Die Methode `RoadSystem.createGroup` hat den Rückgabetyp `Group`.
- Die Methode `RoadSystem.connectConnectors` hat den Rückgabetyp `Connection`.

## **3.2 Einfügen fehlender Signaturen**

In diesem Abschnitt werden fehlende Signaturen aufgeführt, welche im ROSE-Entwurfsdokument vergessen wurden.

### **3.2.1 edu.kit.rose.view.common**

- Die Klasse `ExitSegmentView` wurde hinzugefügt.

### **3.2.2 edu.kit.rose.view.window**

- Die Klasse `ShortCutHelpWindow` wurde hinzugefügt.
- Die Klasse `RoseMenuBar` wurde hinzugefügt.

### **3.2.3 edu.kit.rose.controller.attribute**

- Der Konstruktor von `RoseAttributeController` hat einen Parameter von Typ `ApplicationDataSystem`.

### **3.2.4 edu.kit.rose.controller.command**

- Das Interface `ChangeCommandBuffer` hat die Methode `clear`.

### **3.2.5 edu.kit.rose.controller.hierarchy**

- Das Interface `HierarchyController` hat die Methoden `toggleSegmentSelection`, `addSegmentSelection`, `removeSegmentSelection`, `clearSegmentSelection`, `addSubscription` und `removeSubscription`. Sie ermöglichen es, dass der Nutzer Segmente im `HierarchyPanel` selektieren kann.

### **3.2.6 edu.kit.rose.controller.roadsystem**

- Das Interface `RoadSystemController` hat die Methoden `rotateSegment`, `deleteStreetSegment`, `putSegmentSelection`, `clearSegmentSelection` und `getIntersectionDistance`  
.
- Die Klasse `RotateSegmentCommand` wurde hinzugefügt.

- Das Interface `SelectionBuffer` hat die Methoden `removeAllSelections` und `getSelectedSegments`.

### 3.2.7 `edu.kit.rose.controller.plausibility`

- Das Interface `PlausibilityController` hat die Methode `deleteAllCompatibilityCriteria`.

### 3.2.8 `edu.kit.rose.controller.project`

- Das Interface `ProjectController` hat die Methoden `createNewProject`, `saveAs`, `loadProject`, `loadRecentProject`, `loadBackup` und `getBackupPaths`.

### 3.2.9 `edu.kit.rose.model.roadsystem`

- Das Interface `RoadSystem` hat die Methode `clear`.

## 3.3 Verwendung von JavaFx

In diesem Abschnitt werden Änderungen am ROSE-Enturf aufgeführt, die durch die Verwendung von JavaFx vorgenommen wurden.

### 3.3.1 `edu.kit.rose.infrastructure`

- Die Integer-Parameter aller Methoden der Klassen `Position` und `Movement` sind Double-Parameter.

### 3.3.2 `edu.kit.rose.view.common`

- Die Klasse `SegmentView` hat die Methoden `setupDrag`, `setDraggedConnectorView`, `setOnConnectorViewDrag`, `setOnConnectorViewDragEnd`, `canBeMoved`.

### **3.3.3 edu.kit.rose.view.panel.criterion**

- Die Klasse `CriterionHandle` hat keine `selected`-Eigenschaft. Diese Eigenschaft wurde durch ein JavaFx-SelectionModel ersetzt.
- Die Klasse `CriterionListCell` wurde hinzugefügt.

### **3.3.4 edu.kit.rose.view.panel.hierarchy**

- Die Klasse `ElementView` hat die Methode `onUnmount`.
- Die Klassen `ElementTreeItem` und `ElementTreeCell` wurden hinzugefügt.

### **3.3.5 edu.kit.rose.view.panel.segmentbox**

- Die Klasse `SegmentBoxListCell` wurde hinzugefügt.

## **3.4 Integration von Dependency Injection**

In diesem Abschnitt werden Änderungen am ROSE-Entwurf aufgeführt, die durch die Integration von Dependency Injection vorgenommen wurden.

### **3.4.1 edu.kit.rose.view.common**

- Die Klasse `FxmlContainer` hat die Methode `init`. Sie wird für die Dependency Injection verwendet. Damit entfallen auch die Setter der Kindklassen von `FxmlContainer`, welche Controller oder Model-Objekte setzen.
- Die Klasse `FxmlContainer` hat die Methode `getSubFxmlContainer`.

### **3.4.2 edu.kit.rose.view.window**

- Die Konstruktorparameter der Klassen `RoseWindow`, `MainWindow` und `CriteriaWindow`, welche Controller oder Model-Objekte sind, wurden durch einen Dependency-Injector ersetzt.

- Die Methode `RoseWindow.configureStage` hat einen Parameter vom Typ `Injector`

## 3.5 Verbesserung der Codequalität

In diesem Abschnitt werden Änderungen am Entwurf die zur Verbesserung der Codequalität vorgenommen wurden aufgeführt.

### 3.5.1 `edu.kit.rose.infrastructure`

- Das Interface `Box` hat die Methoden `contains` und `stream`.
- Die Klassen `SimpleUnitObservable`, `SimpleSortedBox`, `SimpleSetObservable`, `SimpleDualSetObservable`, `SimpleBox`, `SimpleApplicationDataSystem`, `SimpleExportStrategy` und `SimpleProject` haben das Präfix "Rose" anstatt dem Präfix "Simple".

### 3.5.2 `edu.kit.rose.view.common`

- Die Eigenschaft "isDraggable" der Klasse `SegmentView` wurde umbenannt zu "isActive".

### 3.5.3 `edu.kit.rose.panel.criterion`

- Die Klasse `CriterionHandle` erweitert `FxmlContainer` anstatt `HBox`.

### 3.5.4 `edu.kit.rose.panel.segment`

- Die Klasse `SelectableAttribute` ist abstrakt.
- Die Klasse `BooleanAttribute` wurde hinzugefügt.

### 3.5.5 `edu.kit.rose.panel.violation`

- Die Klasse `Problem` wurde umbenannt zu `ViolationHandle`.

- Die Klasse `ProblemOverviewPanel` wurde umbenannt zu `ViolationOverviewPanel`

### 3.5.6 `edu.kit.rose.controller.attribute`

- Die Methode `getSharedAttributeAccessors` liegt nicht mehr im `RoadSystem`, sondern im `AttributeController` und wurde umbenannt zu `getBulkEditAccessors`.

### 3.5.7 `edu.kit.rose.controller.command`

- Die Methode `ChangeCommandBuffer.addCommand` wurde durch die Methode `ChangeCommandBuffer.addAndExecuteCommand` ersetzt.

### 3.5.8 `edu.kit.rose.controller.common`s

- Der Konstruktor der abstrakten `Controller`-Klasse hat keinen `ChangeCommandBuffer`-Parameter.

### 3.5.9 `edu.kit.rose.controller.plausibility`

- Im Interface `PlausibilityController` wurde die Methode `setCompatibilityCriterionOperatorType` durch die Methode `setCompatibilityCriterionValidationType` ersetzt.

### 3.5.10 `edu.kit.rose.view.panel.roadsystem`

- Die auf dem `Grid` anzuzeigende `SelectionBox` wird in einer eigenen `SelectionBox`-Klasse implementiert.

### 3.5.11 `edu.kit.rose.model`

- Die Methode `ExportStrategy.exportToFile` ist als abstrakt gekennzeichnet.
- Das Enum `ExportFormat` wurde zu `ProjectFormat` umbenannt.

- Das Interface `ImportStrategy` und die Klasse `RoseImportStrategy` wurden entfernt und durch die Methode `RoseExportStrategy.importToProject` ersetzt.

### **3.5.12 edu.kit.rose.model.roadsystem.elements**

- Die Klasse `Base` hat einen zweiten Konstruktor mit einem String-Parameter.

## **3.6 Änderungen durch Implementierungskontext**

In diesem Abschnitt werden Änderungen am Entwurf aufgeführt, die durch mehr *Kontextinformationen* während der Implementierung entstanden sind.

### **3.6.1 edu.kit.rose.infrastructure**

- Das Interface `Observable` hat die Methode `getThis`.

### **3.6.2 edu.kit.rose.view.common**

- Die `Searchbar` implementiert das Interface `UnitObservable<String>` nicht.
- Die `Searchbar` hat die Methode `searchStringProperty`.
- Die Klassen `EnumLocalizationUtility`, `ConnectorObserver`, `FxmlUtility`, `UnmoutUtility`, `ConnectionView` und `ConnectorView` wurden hinzugefügt.

### **3.6.3 edu.kit.rose.view.panel.hierarchy**

- Die Klasse `ElementeView` implementiert das Interface `SetObserver<Element, Element>` anstatt dem Interface `UnitObserver<Element>`.

### **3.6.4 edu.kit.rose.view.panel.segmentbox**

- Die Klasse `SegmentBluePrint` erweitert `StackPane` anstatt `Pane`.

### **3.6.5 edu.kit.rose.view.panel.violation**

- `MessageFactory` implementiert das Interface `Consumer<Language>` nicht mehr.
- Der Konstruktor der Klasse `ViolationHandle` hat einen Parameter vom Typ `MessageFactory`

### **3.6.6 edu.kit.rose.controller.navigation**

- Die Enums `ErrorType`, `FileDialogType` und `FileFormat` wurden hinzugefügt.
- Die Methode `Navigator.showFileDialog` hat einen Parameter vom Typ `FileDialogType` und einen Parameter vom Typ `FileFormat`.
- Das Interface `Navigator` hat die Methode `showErrorDialog`.

### **3.6.7 edu.kit.rose.controller.project**

- Das Interface `ProjectController` hat die Methode `shutDown`.

### **3.6.8 edu.kit.rose.model**

- Das Interface `Project` hat die Methode `reset`.
- `ApplicationDataSystem` hat die Methoden `addShownAttributeType`, `removeShownAttributeType`, `getRecentProjectPaths` und `addRecentProjectPath`.
- `ApplicationDataSystem` erweitert `DualSetObservable<AttributeType, Path, ApplicationDataSystem>` anstatt `UnitObservable<ApplicationDataSystem>`.
- `RoseApplicationDataSystem` erweitert `RoseSetObservable<AttributeType, ApplicationDataSystem>` anstatt `SimpleUnitObservable<ApplicationDataSystem>`.
- Die Klassen `TimeSliceSetting` und `ZoomSetting` haben die Methode `reset`.

### **3.6.9 edu.kit.rose.model.roadsystem**

- Das Interface RoadSystem hat die Methode `getElementsByName` nicht mehr.
- Das Interface RoadSystem hat die Methode `reset`.

### **3.6.10 edu.kit.rose.model.roadsystem.attributes**

- Der Eintrag NAME im Enum AttributeType wurde entfernt.

### **3.6.11 edu.kit.rose.model.roadsystem.elements**

- Die Klasse Connection hat die Methode `getCenter`.

## **4 Abdeckung der Muss/Wunschkriterien und Anforderungen**

Eines der wichtigsten Ziele der Implementierungsphase war es so viele Muss- und Wunschkriterien wie möglich zu implementieren. Bei dieser Aufgabe haben wir uns gut geschlagen. Im Folgenden werden alle umgesetzten, sowie darunter alle nicht implementierten Kriterien aufgelistet.

### **4.1 Umgesetzte Musskriterien**

- Der Nutzer kann Straßensegmente auf der Hintergrundfläche platzieren, verschieben, verbinden, löschen und auswählen.
- Die Hintergrundfläche der Editorfläche ist verschiebbar, verkleinerbar und vergrößerbar.
- Der Nutzer kann Attribute von Straßensegmenten einsehen und modifizieren.
- Bearbeitungsschritte können rückgängig gemacht und wiederhergestellt werden.
- Der Nutzer kann Kompatibilitätskriterien anlegen, konfigurieren, löschen, importieren und exportieren.
- Der Nutzer kann Verstöße gegen Plausibilitätskriterien einsehen.
- Der Nutzer kann Projekte speichern und laden.
- Der Nutzer kann Straßennetze in das FREEVAL-Format exportieren.
- Der Nutzer kann die Darstellungssprache der Anwendung wechseln.

Damit haben wir alle Musskriterien umgesetzt.

### **4.2 Umgesetzte Wunschkriterien**

- Der Nutzer kann von angezeigten Verstößen gegen Plausibilitätskriterien direkt zu deren Ursache springen.
- Das Programm sichert den aktuellen Bearbeitungsstand automatisch und regelmäßig.

- Der Nutzer kann das Programm über ein Shortcut-System bedienen.

### **4.3 Abgedeckte Funktionale Anforderungen**

Für weitere Informationen siehe Pflichtenheft (Abteilungen: Funktionale Anforderungen, Produktdaten, Nichtfunktionale Anforderungen).

#### **4.3.1 Das Programm kann Straßennetze graphisch darstellen.**

/FA1010M/ Der *Nutzer* kann die *Hintergrundfläche* in der *Editorfläche* verschieben.

/FA1020M/ Der *Nutzer* kann die *Hintergrundfläche* in der *Editorfläche* vergrößern und verkleinern

/FA1030M/ Das *Programm* kann *Straßensegmente* auf der *Hintergrundfläche* darstellen.

/FA1040M/ Der *Nutzer* kann die *Attribute* eines *Straßensegmentes* einsehen.

#### **4.3.2 Der Nutzer kann das Straßennetz aus §4.3.2 bearbeiten.**

/FA2010M/ Das *Programm* zeigt dem *Nutzer* alle verfügbaren *Straßensegment*-Typen in einem *Segmentkasten* an.

/FA2020M/ Der *Nutzer* kann *Straßensegmente* erstellen.

/FA2030W/ Der *Nutzer* kann *Straßensegmente* mit *Drag and Drop* aus dem *Segmentkasten* auf die *Editorfläche* ziehen.

/FA2040M/ Der *Nutzer* kann die *Attribute* von einzelnen *Straßensegmenten* ändern.

/FA2050M/ Bei *Basisstraßensegmenten* kann der *Nutzer* die Enden einzeln (also unabhängig voneinander) verschieben.

/FA2060M/ Der *Nutzer* kann die Enden von zwei *Straßensegmenten* verbinden, indem er sie aneinanderschiebt. (mit /FA2050M/ und /FA2080M/)

/FA2065M/ Der *Nutzer* kann verbundene *Straßensegmente* voneinander trennen.

/FA2070M/ Der *Nutzer* kann mehrere *Elemente* gleichzeitig selektieren.

/FA2080M/ Der Nutzer kann selektierte Straßensegmente (/FA2070M/) auf der Hintergrundfläche verschieben.

/FA2090M/ Der Nutzer kann selektierte Elemente (/FA2070M/) löschen.

/FA2100W/ Der Nutzer kann selektierte Elemente (/FA2070M/) duplizieren.

/FA2110M/ Der Nutzer kann seine letzten Bearbeitungsschritte rückgängig machen und wiederherstellen (“Undo / Redo”).

/FA2120W/ Der Nutzer kann Gruppen erstellen.

/FA2130W/ Der Nutzer kann Elemente maximal einer Gruppen hinzufügen.

/FA2140W/ Der Nutzer kann alle in einer Gruppe indirekt enthaltenen Elemente gemeinsam selektieren.

/FA2150W/ Der Nutzer kann auf der Hintergrundfläche platzierte Straßensegmente drehen.

#### **4.3.3 Das Programm prüft das Straßennetz auf Plausibilität.**

/FA3010M/ Das Programm prüft, ob alle nicht optionalen Attribute der platzierten Straßensegmente vollständig konfiguriert sind. (siehe /PD120/)

/FA3020M/ Das Programm prüft, ob die eingetragenen Werte bei den Attributen der Straßensegmente in den definierten Wertebereichen liegen.

/FA3030M/ Das Programm prüft, ob alle Kompatibilitätskriterien eingehalten werden.

/FA3040M/ Der Nutzer kann eigene Kompatibilitätskriterien anlegen, konfigurieren und löschen.

/FA3050M/ Der Nutzer kann Kompatibilitätskriterien aus einer Kriteriendatei importieren und in eine Kriteriendatei exportieren.

/FA3060M/ Der Nutzer kann eine Liste aller aktuellen Verstöße gegen Plausibilitätskriterien einsehen. (“Problemübersicht”)

/FA3070W/ Der Nutzer kann von einem gewählten Verstoß (/FA3070M/) in der Problemübersicht automatisch zur betroffenen Stelle im Straßennetz springen.

#### **4.3.4 Der Nutzer kann das Projekt exportieren.**

/FA4010M/ Der *Nutzer* kann Speicherort und Dateiformat der Exportdatei dabei auswählen.

/FA4040M/ Das *Programm* kann *Straßennetze* in das *FREEVAL-Format* exportieren.

### **4.4 Der Nutzer kann Projekte speichern und laden.**

/FA5010M/ Der *Nutzer* kann das aktuelle *Projekt* in einer von ihm gewählten *Projektdatei* speichern.

/FA5020M/ Der *Nutzer* kann ein *Projekt* aus einer von ihm gewählten *Projektdatei* laden.

/FA5030W/ Das *Programm* legt automatisch und regelmäßig *Sicherungskopien* vom aktuellen *Projekt* an.

/FA5040W/ Der *Nutzer* kann die *Sicherungskopie* aus /FA5030W/ wieder in das *Programm* laden.

#### **4.4.1 Weitere Anforderungen**

/FA6010M/ Der *Nutzer* kann zur Laufzeit die von der Benutzeroberfläche verwendete Sprache einstellen.

/FA6020W/ Der *Nutzer* kann Teile des *Programms* durch Tastenkombinationen bedienen. Für ein Einblick auf verwendete Tastenkombinationen siehe Pflichtenheft.

/FA6040W/ Der *Nutzer* kann eine Liste aller Tastenkombinationen einsehen.

/FA6050W/ Der *Nutzer* kann *Elemente* anhand ihres Namens suchen.

## 4.5 Abgedeckte Nichtfunktionale Anforderungen

/NF100M/ Das *Programm* muss um neue *Straßensegment*-Typen erweiterbar sein.

/NF110M/ Die *Straßensegment*-Typen müssen um neue *Attribute* erweiterbar sein.

/NF120M/ Das *Programm* muss um neue Exportformate erweiterbar sein.

/NF130M/ Die Sprachen Deutsch und Englisch werden unterstützt. (siehe /FA6010M/)

/NF140M/ Die *Hintergrundfläche* bietet genug Platz, um im Editor mindestens 200 *Straßensegmente* platzieren zu können, ohne dass diese sich überschneiden müssen.

/NF150M/ Das *Programm* soll bei *Straßennetzen* mit bis zu 200 *Elemente* eine maximale *Antwortzeit* von 250 ms haben.

/NF160W/ Das *Programm* kann bis zu 10 *Plausibilitätskriterien* bei bis zu 200 *Straßensegmenten* in unter drei Sekunden auswerten.

/NF170W/ Undo / Redo kann mindestens die letzten 20 Schritte verwalten.

/NF180M/ Die Speicher- und Ladefunktion des *Programms* hat keinen *Informationsverlust* bezüglich des *Projektes*.

/NF190W/ Das *Programm* soll im Falle eines Absturzes maximal fünf Minuten an getätigter Arbeit verlieren.

/NF200M/ Das *Programm* kann unzulässige Eingaben korrekt behandeln.

/NF210M/ Das *Programm* soll um neue Sprachen erweiterbar sein.

## 4.6 Nicht umgesetzte Wunschkriterien

- Der *Nutzer* kann *Straßennetze* in das *SUMO-Format* exportieren.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.
- Der *Nutzer* kann zeitabhängige Messwerte *Straßensegmenten* zuordnen.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft. Außerdem fanden wir, dass unseres Programm durch diese Entscheidung nicht viel von seiner Funktionalität verliert. Allerdings wir haben die Implementierung um diesem Feature erweiterbar gehalten.

## 4.7 Nicht abgedeckte Funktionale Anforderungen

/FA1050W/ Das *Programm* zeigt *Attribute* eines *Straßensegments* auf dem *Straßensegment* an.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.

/FA1060W/ Der *Nutzer* kann die anzugebenden *Attribute* aus /FA1050W/ einstellen.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.

/FA1070M/ Der *Nutzer* kann die *Messwerte* von *Straßensegmenten* einsehen.  
Begründung: /FA2170M/ wurde nicht implementiert.

/FA2160W/ Der *Nutzer* kann gemeinsame Eigenschaften selektierter *Straßensegmente* (/FA2170M/) durch eine einzelne Eingabe modifizieren (“Bulk-Edit”).  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft. Da /FA1070M/, /FA2170M/, /FA2180M/ nicht implementiert wurden, fanden wir, dass ein (“Bulk-Edit”) nicht mehr so viel Mehrwert für den *Nutzer* bietet.

/FA2170M/ Der *Nutzer* kann *Messwerte* von *Straßensegmenten* eingeben und bearbeiten.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft. Außerdem fanden wir, dass unseres Programm durch diese Entscheidung nicht viel von seiner Funktionalität verliert. Allerdings wir haben die Implementierung um diesem Feature erweiterbar gehalten.

/FA2180M/ Der *Nutzer* kann die Länge und Anzahl der *Messwert-Zeitintervalle* konfigurieren.  
Begründung: /FA2170M/ wurde nicht implementiert.

/FA4020M/ Das *Programm* informiert den *Nutzer* über mögliche *Informationsverluste* bzgl. des gewählten Formates beim Export.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.

/FA4030M/ Das *Programm* informiert den *Nutzer* beim Export von *Straßennetzen* über *Verstöße* gegen *Plausibilitätskriterien*, falls es solche gibt.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.

/FA4050W/ Das *Programm* kann *Straßennetze* in das *SUMO-Format* exportieren.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.

/FA6030W/ Das *Programm* führt den *Nutzer* durch eine interaktive Anleitung in ihre Hauptfunktionalität ein.  
Begründung: Diese Anforderung hatte niedrige Priorität für uns.

## **4.8 Nicht abgedeckte Nichtfunktionale Anforderungen**

/NF115M/ Die *Straßensegment*-Typen müssen um neue *Messwerte* erweiterbar sein.  
Begründung: /FA2170M/ wurde nicht implementiert.

/NF220W/ Das *Programm* kann die Breite eines *Straßensegmentes* in Abhängigkeit von der Anzahl an Fahrstreifen darstellen.  
Begründung: Wir haben es innerhalb der gegebenen Zeit nicht geschafft.

## 5 Veränderungen am Implementierungsplan





## 6 Arbeitsabläufe

Ein von uns gestecktes Ziel während der Implementierung von ROSE war es, den Quellcode bereits während der Entwicklung auf die Einhaltung stilistischer Best Practices und ebenso auf Korrektheit zu prüfen.

Wir setzten dieses Ziel durch die Verwendung unterstützender Werkzeuge und durch die Sichtung aller implementierten Funktionseinheiten durch. Zusätzlich nutzten wir automatisierte Werkzeuge, um den geschriebenen Quellcode zu überprüfen: Wir nutzten das Build-Automatisierungs-Tool Gradle, um die Konfiguration der genutzten Entwicklungswerkzeuge zwischen den Teammitgliedern synchron zu halten. Um das Verhalten einzelner Klassen exemplarisch zu überprüfen, nutzten wir Modultests, ein Mocking-Framework und ein Quellcode-Überdeckungswerkzeug. Eine einheitliche Formattierung des Quellcodes wurde durch CheckStyle sichergestellt.

### 6.1 Workflow

Für jede zu implementierende Funktionseinheit des Implementierungsplans legt das zuständige Teammitglied (“Autor”) zunächst einen eigenen Git-Zweig an. Zudem wird direkt ein sogenannter *Merge-Request* in *GitLab* angelegt, in dem die Änderungen von allen Teammitgliedern kommentiert und diskutiert werden können. Der Autor implementiert die Funktion in kleinen Sinneseinheiten und bucht diese regelmäßig im Git-Zweig der Funktionseinheit ein. Sobald der Autor die Implementierung der Funktionseinheit vollendet hat, markiert er den *Merge-Request* als “bereit” und bittet ein oder zwei Personen um eine Durchsicht. Sobald die Durchsicht erfolgreich abgeschlossen ist, können die Änderungen in den Hauptentwicklungszyklus übernommen werden.

### 6.2 Durchsichten

Jede vollständig implementierte Funktionseinheit musste von mindestens einem weiteren Teammitglied als dem Autor gesichtet werden. Bei der Durchsicht wurde geprüft, ob der Quellcode stilistisch unseren Anforderungen entspricht und ob er korrekt und sinnvoll die Spezifikation und den Entwurf implementiert. Dabei wurde insbesondere auf das folgende geachtet:

- Entsprechen die Namen von Variablen, Methoden und Klassen unseren Richtlinien?
- Sind Signaturen und JavaDoc-Kommentare korrekt aus dem Entwurfsdokument übernommen worden?

- Werden Funktionsparameter geeignet validiert?
- Ist der Code angemessen kommentiert und verständlich?
- Für Änderungen am Model- und Controller-Subsystem: Werden alle öffentlichen Methoden und Konstruktoren durch Modultests überprüft?
- Wird in jedem Modultest das Ergebnis angemessen auf Korrektheit geprüft?
- Für Änderungen am View-Subsystem: Können die graphischen Komponenten beim Starten der Anwendung verwendet werden? Entspricht ihre Darstellung den Mockups aus dem Pflichtenheft?

Alle Anmerkungen wurden im *Merge-Request* der Funktionseinheit als Kommentare hinzugefügt, wo sie anschließend diskutiert wurden. Jeder Kommentar musste vom Autor des Quellcodes beantwortet werden, entweder indem er seinen Quellcode rechtfertigte oder indem er die entsprechende Stelle korrigierte. Im Fall einer Korrektur musste das sichtende Teammitglied die geänderten Stellen erneut sichten. Eine Durchsicht war erfolgreich abgeschlossen, sobald alle Kommentare beantwortet wurden.

### 6.3 Entwicklungsumgebung

Alle Projekteinstellungen und Konfigurationen automatisierter Werkzeuge wurden durch das Java-Buildsystem “Gradle” verwaltet. Gradle erlaubte es uns, die Konfiguration der Entwicklungswerkzeuge als Datei in unserem Projekt abzulegen, sodass Änderungen dieser Konfigurationen ebenso wie Änderungen am Quellcode an alle Teammitglieder verteilt wurden. Konkret konfigurierten wir mithilfe von Gradle die Abhängigkeiten unserer Anwendung auf andere Java-Bibliotheken (bspw. JavaFX und JGraphT), die Kompilierung von ROSE und die Ausführung unserer Modultests. Dies erlaubte es uns, auf verschiedenen Betriebssystemen am Projekt zu arbeiten, ohne dass die entsprechenden Tools manuell neu eingerichtet werden mussten. Das Gradle-Projektformat wird zwar von vielen Entwicklungsumgebungen unterstützt, dennoch haben wir uns auf die Verwendung der Entwicklungsumgebung IntelliJ IDEA geeinigt, um Tipps und Tricks bei der Quellcodeeditierung möglichst reibungslos untereinander austauschen zu können.

Darüber hinaus verwendeten wir “Git” als Versionskontrollsystem, da dies der aktuelle Industriestandard ist und die Verwendung von Git allen Teammitgliedern bereits durch die “Softwaretechnik I”-Vorlesung bekannt war. Zur Synchronisation unserer Git-Repositories verwenden wir die *GitLab*-Instanz des SCC. *GitLab* erlaubte uns zudem, den Quellcode zu diskutieren, und ließ sich darüber hinaus als “Continuous-Integration”-Werkzeug verwenden. Dadurch konnten wir unsere automatisierten Prüfwerkzeuge nach jeder Einbuchung in einer virtualisierten Umgebung ausführen. Durch diese automatische Ausführung wur-

den unsere Modultests zu Regressionstests und überprüften stichprobenartig, ob durch Änderungen am Quellcode bestehende Funktionalitäten kaputt gegangen sind.

## 6.4 Modultests

Wir hatten es uns als Ziel gesetzt, bereits während der Implementierung eine breite Überdeckung des Quellcodes durch Modultests zu erreichen. Dies hielten wir für notwendig, da wir es bspw. bei den Straßennetzen und Plausibilitätskriterien mit komplexen Datenstrukturen zu tun haben, die funktional mit nahezu allen Komponenten der Anwendung zusammenhängen.

Dabei beschränkten wir im Rahmen der Implementierungsphase den Fokus der Modultests auf das Model- und auf das Controller-Subsystem, da der notwendige Aufwand für die Implementierung automatisierter Modultests von Benutzeroberflächen für uns zeitlich nicht im Rahmen der Implementierungsphase möglich ist. Für die Qualitätssicherungsphase ist allerdings auch für die Testung des View-Subsystems eine systematischere Herangehensweise geplant.

Paket	Überdeckung	
	Zeilen	Zweige
edu.kit.rose.model.plausibility.criteria	84 %	63 %
edu.kit.rose.controller.roadsystem	61 %	61 %
edu.kit.rose.model.roadsystem.elements	95 %	87 %
edu.kit.rose.model	90 %	66 %
edu.kit.rose.model.roadsystem.measurements	0 %	n/a
edu.kit.rose.model.roadsystem	87 %	56 %
edu.kit.rose.controller.hierarchy	91 %	94 %
edu.kit.rose.controller.plausibility	76 %	60 %
edu.kit.rose.controller.selection	74 %	62 %
edu.kit.rose.controller	73 %	n/a
edu.kit.rose.controller.project	0 %	0 %
edu.kit.rose.controller.attribute	90 %	65 %
edu.kit.rose.infrastructure.language	84 %	50 %
edu.kit.rose.model.plausibility.violation	89 %	87 %
edu.kit.rose.controller.command	88 %	66 %
edu.kit.rose.controller.commons	80 %	67 %
edu.kit.rose.controller.measurement	0 %	n/a
edu.kit.rose.controller.navigation	80 %	n/a
edu.kit.rose.model.roadsystem.attributes	98 %	n/a
edu.kit.rose.controller.application	94 %	50 %
edu.kit.rose.model.plausibility.criteria.validation	100 %	100 %
edu.kit.rose.infrastructure	82 %	69 %
edu.kit.rose.model.plausibility	100 %	100 %

Tabelle 1: Quellcodeüberdeckung durch Modulteste, Paketnamen sind als Unterpakete von `edu.kit.rose` zu verstehen

## 7 Statistiken

In diesem Abschnitt werden die Statistiken zu Arbeitzeit und Codezeilenanzahl vorgestellt. Zur Erhebung dieser Daten wurden während der Implementierungsphase tabellarisch Arbeitszeit und geschriebene Codezeilen erfasst.

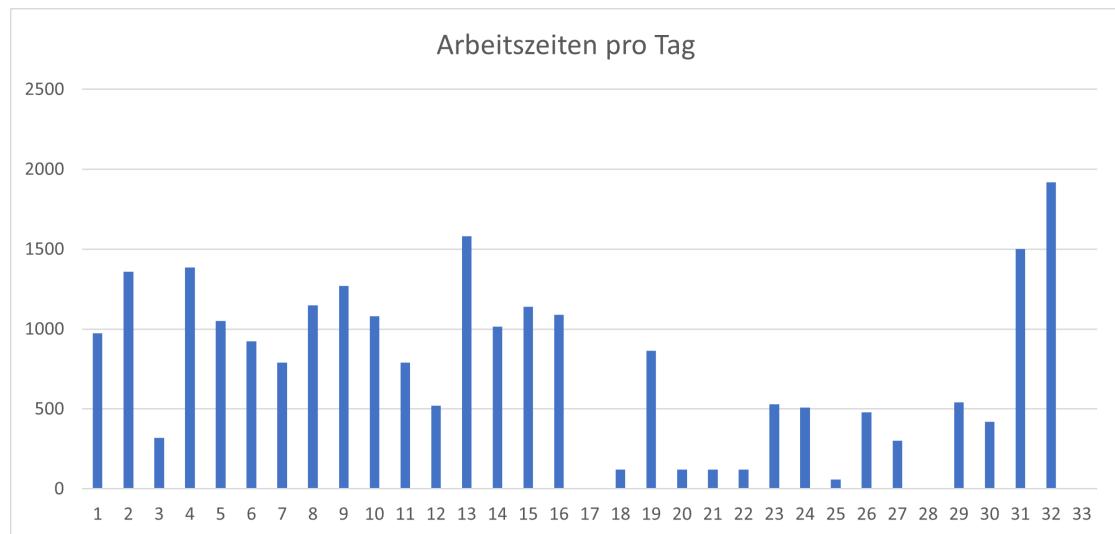


Abbildung 1: Arbeitszeiten pro Tag in Minuten

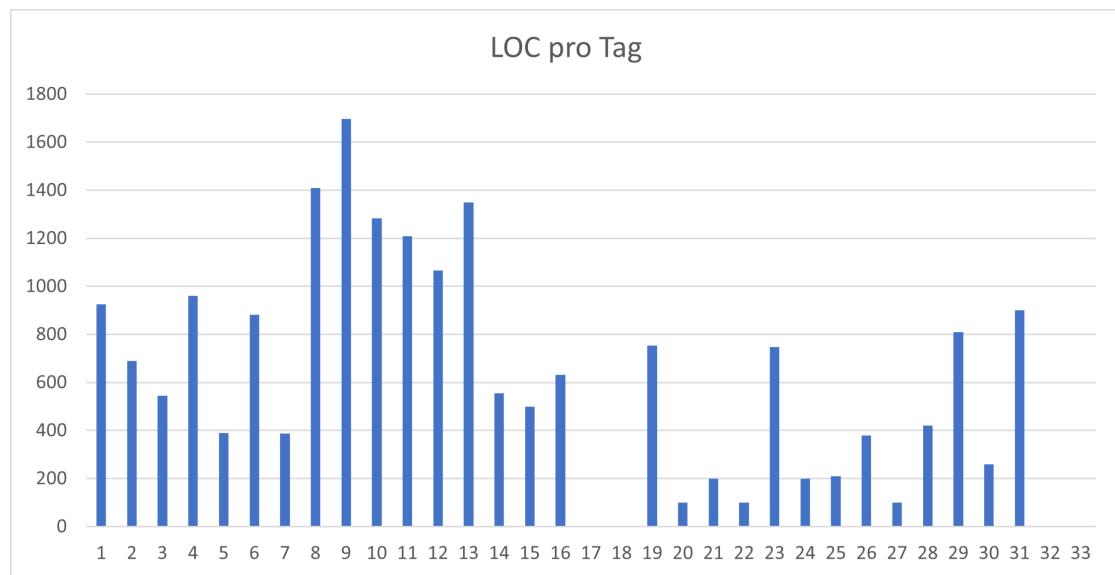


Abbildung 2: Codezeilenanzahl pro Tag

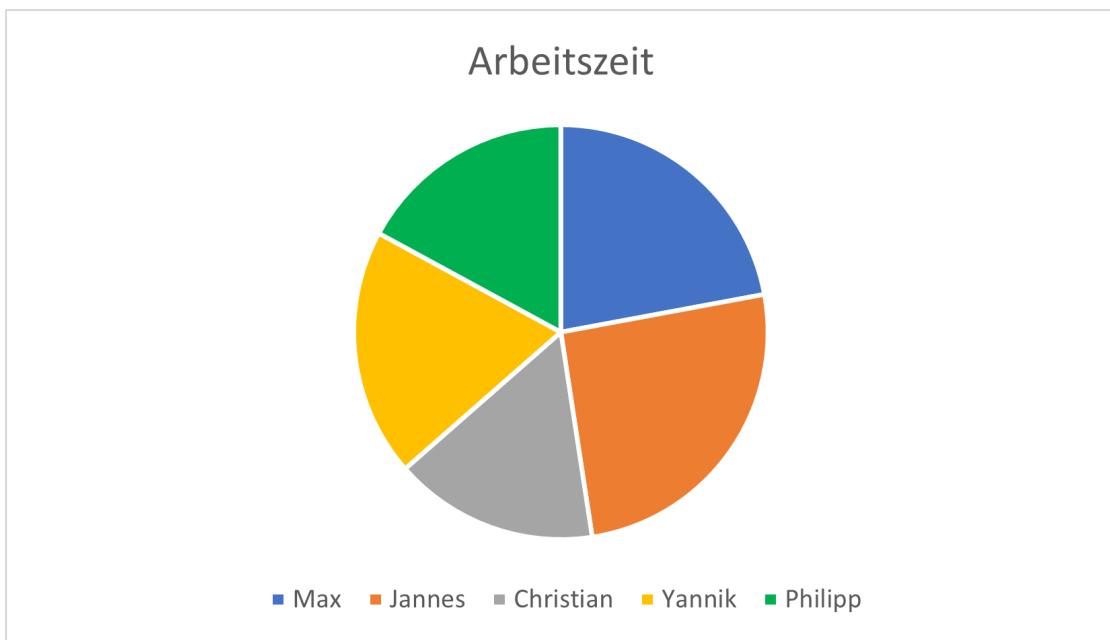


Abbildung 3: Arbeitszeitanteile der Teammitglieder

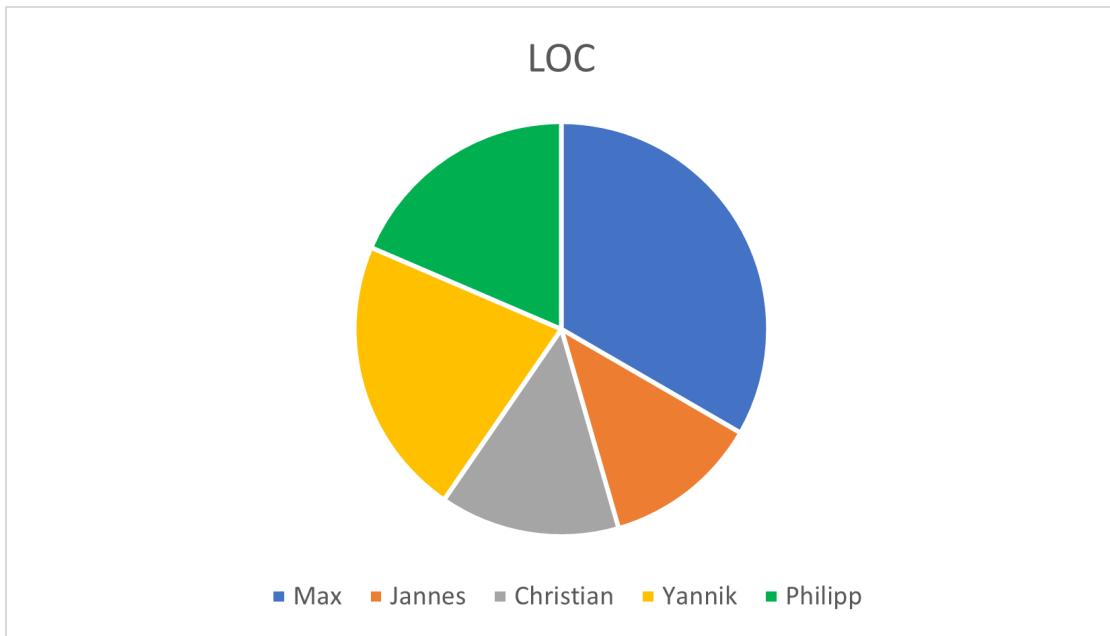


Abbildung 4: Codezeilenanteile der Teammitglieder

## Glossar

**Antwortzeit** Die Zeitspanne zwischen einer Nutzereingabe und der damit verbundenen visuellen Reaktion des *Programms*.

**Attribut** Ein Attribut ist eine Eigenschaft, die einem konkreten *Straßensegment* zugeordnet ist. Der *Nutzer* kann einen Wert für ein Attribut festlegen.

**Basisstraßensegment** Ein *Straßensegment*, das genau eine Fahrtrichtung und 2 Enden hat, bei dem man die Enden unabhängig voneinander bewegen kann.

**direkt enthalten** Ein *Element* A ist in einer Gruppe B direkt enthalten, wenn A indirekt in B enthalten ist und es in B keine weitere Gruppe C gibt, sodass C A indirekt enthält.

**Drag and Drop** Der Nutzer kann *Elemente* aufheben, indem er den Mauszeiger darüber bewegt und die linke Maustaste drückt. Mit weiterhin gedrückter linker Maustaste kann er das *Element* nun verschieben. Das *Element* wird an der Mauszeigerposition abgelegt, sobald der *Nutzer* die linke Maustaste loslässt.

**Editorfläche** Die Editorfläche ist ein Bedienelement auf dem ein Ausschnitt des *Straßen-*  
*netzes* und der *Hintergrundfläche* dargestellt wird.

**Element** Ein Element ist entweder ein *Straßensegment* oder eine *Gruppe*. Es kann in maximal einer *Gruppe* direkt enthalten sein. Ein Element ist in allen *Gruppen* die höher in der Gruppenhierarchie stehen indirekt enthalten.

**FREEVAL-Format** Ein am Institut für Verkehrswesen am KIT spezifiziertes Format für den Import von *Straßennetzen* in die Software “Deutsches FREEVAL”. Das Format gibt eine bestimmte Struktur für YAML-Dokumente vor. Die Spezifikation kann im Anhang §?? eingesehen werden.

**GitLab** GitLab ist eine Webanwendung zur Verwaltung von Git-Repositories. Für die Implementierung von ROSE wird eine vom KIT bereitgestellte GitLab-Instanz genutzt.

**Gruppe** Gruppen sind *Elemente*, die weitere *Elemente* enthalten können. Der *Nutzer* kann Gruppen verwenden, um logisch zusammengehörende *Elemente* einander zuzuordnen. Der Nutzer kann *Elemente* willkürlich Gruppen zuordnen. Gruppen können nicht in sich selbst enthalten sein.

**Hintergrundfläche** Die Hintergrundfläche ist eine Fläche mit angezeigtem Raster, auf dem der Benutzer seine *Straßensegmente* ablegt und sein *Straßennetz* editiert. Ein Ausschnitt der Hintergrundfläche wird auf der *Editorfläche* angezeigt.

**indirekt enthalten** Ein *Element* ist in allen *Gruppen* die höher in der Elementhierarchie stehen indirekt enthalten.

**Informationsverlust** Bei der Speicher- bzw. Exportfunktion liegt ein Informationsverlust vor, wenn nicht alle Daten des *Projektes* bzw. des *Straßennetzes* im Zielformat gespeichert werden. Beispielsweise kann beim Export in das *FREEVAL-Format* nicht die Positionierung der *Straßensegmente* auf der *Hintergrundfläche* gespeichert werden.

**Kompatibilitätskriterium** Ein Kompatibilitätskriterium definiert eine Beziehung zweier Werte für ein bestimmtes *Attribut*, das von allen verbundenen *Straßensegmenten*, einer spezifizierten Menge von Segmenttypen), erfüllt sein muss. Sie werden vom *Nutzer* spezifiziert.

**Kontextinformationen** Kontextinformationen sind Informationen über die Codestruktur, über benötigte oder nicht benötigte Codeelemente. Solche Codeelemente können z.B. Methodenparameter, Methoden oder Klassen sein.

**Kriteriendatei** Eine Kriteriendatei ist eine Datei die Definitionen von *Plausibilitätskriterien* enthält. Sie kann in das *Programm* importiert werden und aus dem *Programm* exportiert werden.

**Merge-Request** Ein Merge-Request wird in *GitLab* erstellt, um einen Git-Zweig in den Hauptzweig einzugliedern. Er kann genutzt werden, um die im Nebenzweig hinzugefügten Änderungen zu diskutieren.

**Messwert** Ein Messwert ist ein vom Nutzer eingetragener Wert, der vom Aufbau eines *Straßennetzes* unabhängig ist. Messwerte werden in *Zeitschritten* organisiert. Messwerte können mit einem *Straßennetz* exportiert werden, falls sie vom Exportformat unterstützt werden.

**Messwertübersicht** Die Messwertübersicht ist eine Ansicht, in welcher der *Nutzer* die *Messwerte* aller *Straßensegmente* eintragen und bearbeiten kann.

**Nutzer** Endnutzer des *Programms*, Einzelperson die eine Lizenz für die Nutzung des *Programms* besitzt.

**optional** Ein *Attribut* ist optional, wenn der Export des *Straßennetzes* auch mit nicht spezifiziertem Wert möglich ist.

**Plausibilitätskriterium** Ein Plausibilitätskriterium ist eine Eigenschaft die ein *Straßensegment* erfüllen muss. Es handelt sich dabei entweder um ein *Kompatibilitätskriterium*, ein *Vollständigkeitskriterium* oder um ein *Wertebereichskriterium*.

**Problemübersicht** Der Bereich der graphischen Benutzeroberfläche, in dem die Verstöße gegen *Plausibilitätskriterien* angezeigt werden.

**Programm** Das hier zu entwickelnde Programm: ROSE (Road System Editor).

**Projekt** Ein Projekt umfasst genau ein *Straßennetz*. Ein Projekt kann in einer *Projektdatei* gespeichert werden. Der *Nutzer* kann ein Projekt im *Programm* öffnen, indem er die entsprechende *Projektdatei* lädt.

**Projektdatei** Eine Projektdatei ist eine Datei, in der alle Daten eines *Projekts* gespeichert werden. Das *Programm* kann Projektdateien speichern und laden. Das Laden einer Projektdatei entspricht dem Öffnen eines *Projekts*.

**Segmentkasten** Der Segmentkasten ist ein Bedienelement der graphischen Benutzeroberfläche. Es listet dem *Nutzer* alle verfügbaren *Straßensegment-Typen* auf und gibt ihm die Möglichkeit, *Straßensegmente* zu erstellen.

**selektiert** Ein *Element* oder ein *Plausibilitätskriterium* ist dann selektiert, wenn es mit einem Mausklick angeklickt wurde.

**Sicherungskopie** Eine Sicherungskopie ist eine Kopie einer *Projektdatei*, die dazu dient, den Zustand eines *Projektes* zu einem bestimmten Zeitpunkt zu sichern.

**Straßennetz** Ein Straßennetz besteht aus einem, oder mehreren *Straßensegmenten* und allen Verbindungen zwischen ihnen.

**Straßensegment** Ein Straßensegment repräsentiert einen Straßenabschnitt. Straßensegmente haben Endpunkte, an die andere Straßensegmente mit deren Endpunkten angeschlossen werden können. *Straßensegmente* können *Basisstraßensegmente*, Einfahrten oder Ausfahrten sein. So kann der *Nutzer* Straßensegmente zu einem *Straßennetz* verbinden. Straßensegmente sind *Elemente*.

**SUMO-Format** Das SUMO-Format ist ein Dateiformat der Verkehrssimulationssoftware SUMO.

**Verstoß** Ein Widerspruch zu einem *Plausibilitätskriterium*.

**Vollständigkeitskriterium** Ein Vollständigkeitskriterium definiert, welche *Attribute* eines *Straßensegments* für den Export in ein gegebenes Format notwendig sind.

**Wertebereichkriterium** Wertebereichkriterien beschreiben die Wertebereiche in denen *Attribut*-Werte liegen dürfen.

**Zeitschritt** Ein Zeitschritt ist eine Zeitspanne, deren Länge und Anzahl in der *Messwertübersicht* vom *Nutzer* festgelegt wird.