Driving Innovation in Crisis Management for European Resilience

D923.11 - FUNCTIONAL SPECIFICATION OF THE TEST-BED INFRASTRUCTURE

SP92 - TEST-BED

MARCH 2018 (M47)

This project has received funding from the European Union's 7th Framework Programme for Research, Technological Development and Demonstration under Grant Agreement (GA) N° #607798

Project information

Project Acronym:	DRIVER+
Project Full Title:	Driving Innovation in Crisis Management for European Resilience
Grant Agreement:	607798
Project Duration:	72 months (May 2014 - April 2020)
Project Technical Coordinator:	TNO
Contact:	coordination@projectdriver.eu

Deliverable information

Deliverable Status:	Final
Deliverable Title:	D923.11 - Functional specification of the Test-bed infrastructure
Deliverable Nature:	Report (R)
Dissemination Level:	Public (PU)
Due Date:	March 2018 (M47)
Submission Date:	03/04/2018
Sub-Project (SP):	SP92 - Test-bed
Work Package (WP):	WP923 – Test-bed Infrastructure
Deliverable Leader:	XVR Simulation
Reviewers:	Hector Naranjo – GMV, Michael Middelhoff – WWU, Angela Schmitt – DLR, Marcel van Berlo - TNO
File Name:	DRIVER+_D923.11_Functional Specification of the Test-bed.docx

Revision Table

Issue	Date	Comment	Author
V0.1	08/11/2017	Initial draft	Steven van Campen, XVR Sub-Task Leader 923.1.2
V0.2	20/11/2017	Split in multiple documents, this one being the main, high-over specifications document. And all specs documentation done in GitHub.	Steven van Campen, XVR Chiara Fioni, JRC Erik Vullings, TNO
V0.3	23/03/2018	Version published and reviewed on GitBook	Steven van Campen, XVR Erik Vullings, TNO Martijn Hendriks, XVR Joaquin Marquez-Bugella, FRQ Cyril Dangerville, TS Hector Naranjo, GMV Michael Middelhoff, WWU Angela Schmitt, DLR Chiara Fonio, JRC
V0.4	26/03/2018	GitBook version transferred to Word file	Steven van Campen, XVR
V0.5	28/03/2018	Included final comments by JRC and GMV	Steven van Campen, XVR Chiara Fonio, JRC Hector Naranjo, GMV Marcel van Berlo, TNO
V0.6	30/03/2018	Final draft approved	Steven van Campen, XVR Marcel van Berlo, TNO
V0.7	03/04/2018	Final check and approval for submission	Peter Petiet, Project Director, TNO
V1.0	03/04/2018	Final check and submission to the EC	Francisco Gala, ATOS

The DRIVER+ project

Current and future challenges due to increasingly severe consequences of natural disasters and terrorist threats require the development and uptake of innovative solutions that are addressing the operational needs of practitioners dealing with Crisis Management. DRIVER+ (Driving Innovation in Crisis Management for European Resilience) is a FP7 Crisis Management demonstration project aiming at improving the way capability development and innovation management is tackled. DRIVER+ has three main objectives:

1. Develop a pan-European Test-bed for Crisis Management capability development:
   - Develop a common guidance methodology and tool (supporting Trials and the gathering of lessons learned.
   - Develop an infrastructure to create relevant environments, for enabling the trialling of new solutions and to explore and share CM capabilities.
   - Run Trials in order to assess the value of solutions addressing specific needs using guidance and infrastructure.
   - Ensure the sustainability of the pan-European Test-bed.
2. Develop a well-balanced comprehensive Portfolio of Crisis Management Solutions:
   - Facilitate the usage of the Portfolio of Solutions.
   - Ensure the sustainability of the Portfolio of Solutions.
3. Facilitate a shared understanding of Crisis Management across Europe:
   - Establish a common background.
   - Cooperate with external partners in joint Trials.
   - Disseminate project results.

In order to achieve these objectives, five Subprojects (SPs) have been established. **SP91** ***Project Management*** is devoted to consortium level project management, and it is also in charge of the alignment of DRIVER+ with external initiatives on crisis management for the benefit of DRIVER+ and its stakeholders. In DRIVER+, all activities related to Societal Impact Assessment (from the former SP8 and SP9) are part of SP91 as well. **SP92** ***Test-bed*** will deliver a Guidance methodology and guidance tool supporting the design, conduct and analysis of Trials and will develop a reference implementation of the Test-bed. It will also create the scenario simulation capability to support execution of the Trials. **SP93** ***Solutions*** will deliver the Portfolio of Solutions (PoS) which is a database driven web site that documents all the available DRIVER+ solutions, as well as solutions from external organizations. Adapting solutions to fit the needs addressed in Trials, will be done in SP93. **SP94** ***Trials*** will organize four series of Trials as well as the final demo. **SP95** ***Impact, Engagement and Sustainability***, is in charge of communication and dissemination, and also addresses issues related to improving sustainability, market aspects of solutions, and standardization.

The DRIVER+ Trials and the Final Demonstration will benefit from the DRIVER+ Test-bed, providing the technological infrastructure, the necessary supporting methodology and adequate support tools to prepare, conduct and evaluate the Trials. All results from the trails will be stored and made available in the Portfolio of Solutions, being a central platform to present innovative solutions from consortium partners and third parties and to share experiences and best practices with respect to their application. In order to enhance the current European cooperation framework within the Crisis Management domain and to facilitate a shared understanding of Crisis Management across Europe, DRIVER+ will carry out a wide range of activities, whose most important will be to build and structure a dedicated Community of Practice in Crisis Management, thereby connecting and fostering the exchange on lessons learned and best practices between Crisis Management practitioners as well as technological solution providers.

Executive summary

This deliverable provides the requirements specifications – meaning both must-do requirements and should/could-do wishes - of the to-be-developed Test-bed within the DRIVER+ project (D+). This Test-bed consists of software components intended to support Trials, which are systematic tests of Crisis Management (CM) solutions in realistic but non-operational (fictive) contexts. Solutions are any type of tool/product/procedure (e.g. a software package, a training method or a new standard operating procedure) that is intended to support/improve Crisis Management. The purpose of conducting Trials in DRIVER+ is to find out if and how some innovative solutions can help resolve the needs of the CM practitioners, before adopting these solutions.

The development of the Test-bed is done within WP923. The Test-bed and its development are:

- Linked to the design and evaluation of the Trial Guidance Methodology in WP922.
- Explained in the Training Module in T924.1.
- Described in the Portfolio of Solutions, developed in WP93.
- Used in the Trials of SP94.
- Made available open-source to support the sustainability of DRIVER+ results in WP954.

The Test-bed will be delivered in 3 versions. Version 1 is intended for use in Trials 1 and 2, and consists of a limited number of components (i.e. Scenario Manager and AAR not included and other components in first prototype quality). Version 2 is to be used in Trials 3 and 4 and the Final Demo and comes with all components with a quality level surpassing that of a first prototype. It should contain data-sets and basic scenarios that can be used for effectively implementing and testing the Test-bed. The final version, Version 3, should have an even better quality based on experiences gathered from the use of the Test-bed in the executed Trials and is made available open-source on GitHub.

This deliverable is available as living document on the online, open-source document-sharing platform GitBook[1], such that it can be updated when development and use of the Test-bed progresses and to open up these specifications to anyone outside DRIVER+. This document describes what the Test-bed must/should be. How these requirements and wishes are implemented is documented in Deliverables D923.21, D923.22 and D923.23 - Reference Implementations of the Test-bed (versions 1/2/3 resp.).

The documented specifications are based on lessons learned from experiments done in the former phase of the project, use-case analysis, experiences from development partners in other projects, market analysis into comparable components and consulting DRIVER+ Trial organizers and Solution providers. The specifications drafting process runs in parallel with the design and development process of the Test-bed's components such that both processes can influence each other positively in a quick and agile manner.

The following lessons were learned from the use of previous Test-beds in the former DRIVER experiments:

- A generic Trial Guidance Methodology is needed for analysing the need, and preparing, executing and reviewing a Trial.
- Only one Test-bed must be created and this must be used for each and every Trial.
- The Test-bed should be open source.
- The Test-bed should have a message-oriented architecture.
- The Test-bed should use well-defined, easily accessible, syntactically correct messages, and close to common standards.
- The Test-bed should be clearly defined and scoped, i.e. what it is, and what it is not.

[1] driver-eu.gitbooks.io/test-bed-specification/content/ [1]

The Test-bed must be easily reproducible, and offer administrative as well as supporting tools and services.

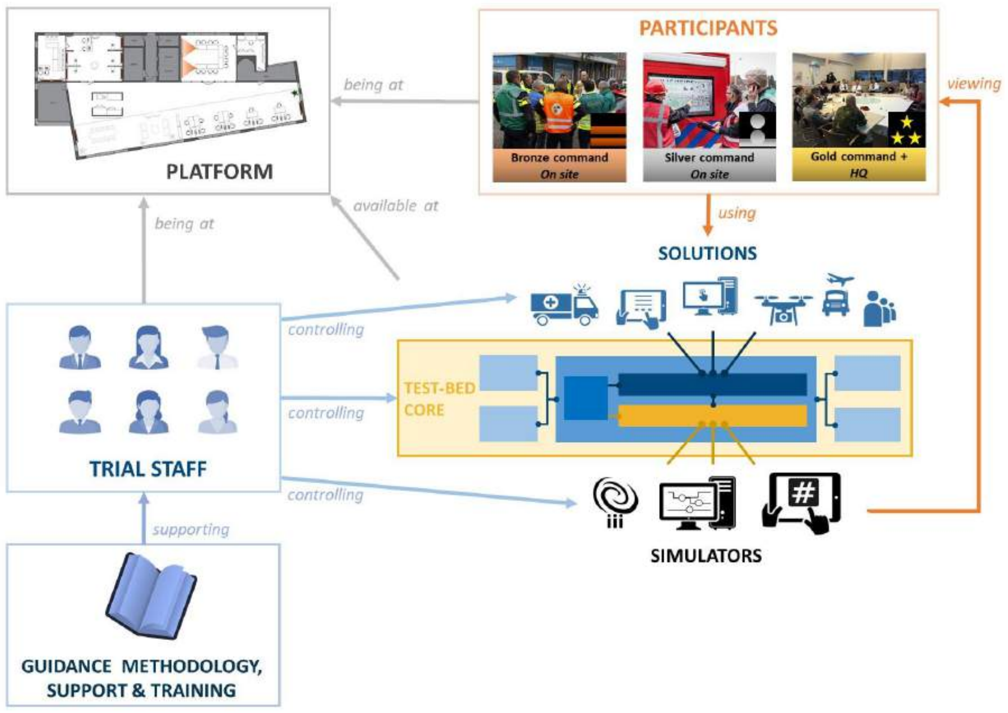

Figure Sum.1: Positioning of the Test-bed

As depicted in Figure Sum.1, the Test-bed will be used in the following environment:

- **Participants** are using the **Solutions** to solve a fictive crisis. Data-integration between these solutions flows via the **Test-bed**.
- This fictive, virtual crisis exists in a virtual world created by a set of **Simulators** (e.g. computer-based simulators, or a fire brigade's physical exercise terrain, or a combination of both). Data-integration between the simulators flows via the **Test-bed**.
- The **Trial Staff**, supported by the **Trial Guidance Methodology** and the **Training Module**, uses the Test-bed during the:
    - *Preparation-phase* – to build the scenario and set-up and test all solutions and simulators required.
    - *Execution-phase* – to control how the virtual crisis scenario evolves, observe the Participants and gather data-logs from both the solutions and the virtual crisis.
    - *Evaluation-phase* – to gather and analyse all observations, recordings and data-logs, and base an evidence-based evaluation report on these.
- The Test-bed, solutions and simulators are available at the **Platform** (i.e. one or more locations), which provides rooms/areas for Participants to execute CM operations and rooms/areas for the Trial Staff. The Platform can be a simulation centre like available in The Hague, but can also be a live exercise terrain.

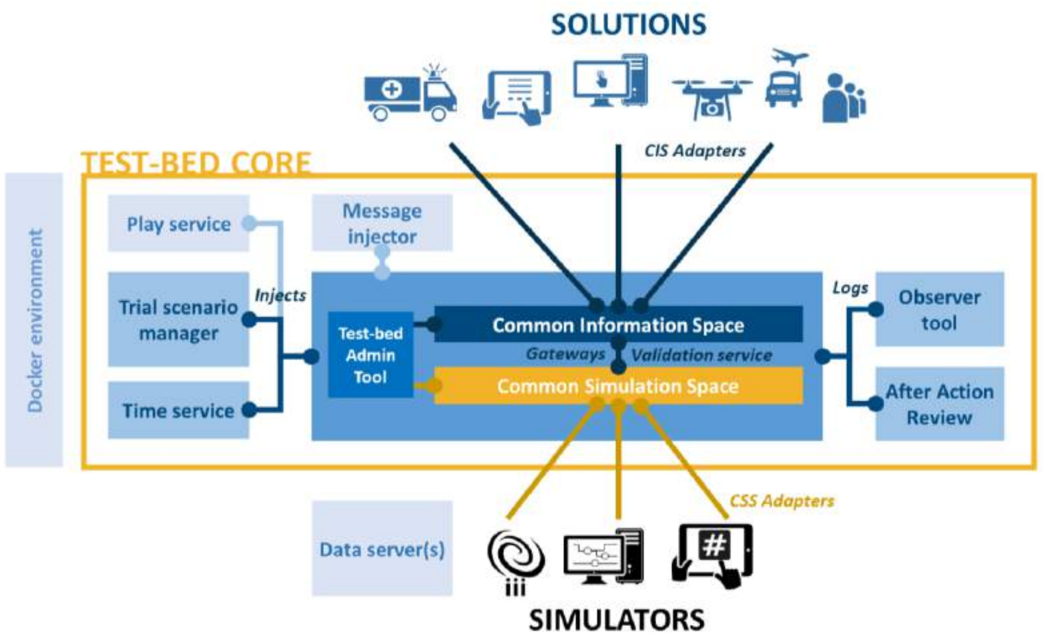

Figure Sum.2: All components available within the Test-bed

To support this use, the Test-bed must consist of the following components (see Figure Sum.2):

- The **Common Information Space (CIS)** is a central messaging bus with standardized CIS Adapters to connect solutions to the Test-bed.
- The **Common Simulation Space (CSS)** is a similar central messaging bus with standardized CSS Adapters to connect simulators to the Test-bed.
- The CIS and CSS are interlinked via one or more **Gateways** to feed solutions with data from the simulated crisis (e.g. positions of virtual police cars driving around in the fictive region) and to pass through changes inputted via one or more solutions to the simulators (e.g. a command to block off an area). A **Validation Service** can be added to safeguard messages are correctly filled and understandable by the counterpart.
- Both the CIS and the CSS are controlled via the **Test-bed Admin Tool**. This provides an interface for Trial Staff to control which solutions and simulators are to be started and which data is (not) exchanged.
- The **Trial Scenario Manager** and the **Time Service** components provide functionality and an interface for Trial Staff to create a high-over timeline of the Trial Scenario with Injects (in the preparation-phase), activate these Injects if needed (in the execution-phase) and control the fictive time clock (in the execution-phase, meaning starting the Trial, pausing the fictive clock, fast-forwarding, jumping back in fictive time and stopping the Trial).
- The **Observer Support Tool (OST)** provides a module to prepare, make and gather observations.
- The **After Action Review (AAR)** module gathers in execution-phase data-logs, observations and (screen-) recordings, such that these can be reviewed.

For development and implementation purposes, the Test-bed should also come with:

- A **Message Injector** to input specific, single messages to test whether these messages are correctly received by another component.
- A **Play Service** to input several messages in chronological order.
- **Data Server(s)** to feed the simulators and potentially also the solutions with geo-specific information and map-layers.
- A **Docker Environment** providing images (i.e. installers) of each component, such that the installation of (a limited) Test-bed can be done quickly and transparently.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AAR	After Action Review
AVRO	Open data serialization system supported by the Apache Organization
CAP	Common Alerting Protocol
CIS	Common Information Space
CM	Crisis Management
COP	Common Operational Picture tools for creating a shared situational awareness
CoPCM	Community of Practice in Crisis Management
COW	Common Online Workspace
CSS	Common Simulation Space
C#	[C sharp] Programming Language
C++	[C plus plus] Programming Language
Docker	Container environment to enable independence between applications and infrastructure
DOW	Description Of Work
D[xxx]	Deliverable number xxx
EDXL	Emergency Disaster eXchange Language
EMSI	Emergency Management Shared Information
FRQ	Frequentis (DRIVER+ partner)
GitBook	Open Source service for creating online books
GitHub	Repository for managing DRIVER+'s software code
GT	Guidance Tool
GUI	Graphical User Interface
Java	Programming Language
Kafka	An open source distributed streaming platform supported by the Apache Organization that is used as the basis to exchange information between simulators and solutions
KPI	Key Performance Indicator
MBtiles	Single file database format to store images of a map
MGT	Management
OST	Observer Support Tool
Python	Programming Language

Acronym	Definition
REST	Representational State Transfer (common interface allowing you to read and write data from a service)
SP	Sub-project
SUMO	Simulation of Urban Mobility, a traffic and pedestrian simulator
TAP	Trial Action Plan
TGM	Trial Guidance Methodology
TCS	Thales Services (DRIVER+ partner)
TypeScript	Programming Language
WFS	Web Feature Service (serving a map layer as vectors)
WMS	Web Mapping Service (serving a map layer as picture)
WP	Work package
XACML	eXtended Access Control Mark-up Language, a standard for describing security permissions to resource
XML	eXtended Mark-up Language, a textual representation of a message that is easily readable by computers

1. Introduction

This document is intended to serve as functional requirements specifications for the DRIVER+ Test-bed, meaning both requirements (must-dos) and wishes (should-dos) are listed. This Test-bed consists of software components, developed within Work-Package 923 and intended to support Trials, which are systematic tests of Crisis Management (CM) solutions in realistic but non-operational (fictive) contexts. Solutions are any type of tool/product/procedure (e.g. a software package, a training method or a new standard operating procedure) that is intended to support/improve Crisis Management. The purpose of conducting Trials in DRIVER+ is to find out if and how innovative solutions can help resolve the needs of the CM practitioners, before adopting these solutions.

This document is linked to D923.21, D923.22 and D923.23 that describe a reference implementation of the released version of the Test-bed's components, including design and development choices taken per component. This document states the functional requirements for these components (i.e. what these components must/should do and why this is needed). How these requirements are fulfilled and design/development choices made during the component's development is documented in the Reference Implementation documentation (D923.21, D923.22 and D923.23). Note that 3 versions of the Test-bed are foreseen, each accompanied with its own Reference Implementation deliverable.

In the following chapters, this document provides:

- The lessons learned from test-beds developed and used in experiments in the former phase of the project (chapter 3).
- The high-over use-cases of the entire, new, to-be-developed DRIVER+ Test-bed in the environment it is used in, in the different phases of use and its different user-types (chapter 4).
- The functional specifications of this Test-bed and the components it therefore exists of (chapter 4 and chapter 5). The latter chapter also provides links for each component to separate, more detailed specification documents of each component.
- The process how these specifications were drafted (chapter 2).
- What can be concluded from the functional specifications of the DRIVER+ Test-bed, the high-over development cycle and the development dependencies of these Test-bed components with regards to other WPs and SPs (chapter 6).

1.1 Targeted reading audience

This document is targeted at readers who know the DRIVER+ objectives and project-contents at least at a high-over level. They should also have a minimal level of understanding of software functional design.

Primary target audiences of this document are:

- Software designers, developers and testers creating a part of the Test-bed and/or wanting to use the Test-bed.
- Trial Committee members (i.e. people involved in the organization of Trials within DRIVER+).
- DRIVER+ Trial Guidance Methodology (TGM) developers and those supporting the implementation of this TGM (see D922.21 - Trial guidance methodology and guidance tool specifications (version 1)).
- DRIVER+ externals, who are interested in what the Test-bed can bring them.

Readers interested in how a specific component functions, the design and development choices made during its development and the way to implement these components, are referred to the Reference Implementation documentation per version of the Test-bed (i.e. D923.21, D923.22 and D923.23).

The core of this deliverable is an online version of it, published on GitBook.com. This online document is a living document, which is updated throughout the development of the Test-bed. It is open to read by anyone.

For editing and reviewing this document, a personal GitHub-account is required with which one can request and be given access to the DRIVER+ GitHub group.

1.2 Online living document on GitBook

The documentation of the Test-bed's functional specifications is completely done in the online developer tool GitBook, a website linked to the popular developer community GitHub in which software code can be created, shared and adjusted. GitBook is a website on which documents can be placed, read and adjusted. For official delivery, a one-time offline copy is made, saved as .doc and .pdf files and delivered end of March 2018.

The decision to use GitBook for creating these functional specifications (i.e. instead of using MS Word for all documentation for D923.11) is based on the considerations that:

GitHub and its affiliative GitBook is targeted specifically for software development

The overall majority of person-months allocated to the Test-bed's development is in fact software development.

Specifications are living documents

All the Test-bed's specification documents are foreseen to be living documents to include progressing knowledge on the Test-bed and practical considerations and decisions made during the development and use of the Test-bed, like common in any software development project. This requires a documentation process and tooling that can deal with these ongoing updates/changes to the specifications.

Parallel, agile development

GitHub and GitBook are developed specifically for parallel (agile) software development and for easy sharing of documents and their updates amongst developers. In the agile software development method, multiple developers work in parallel on the same/linked software components and the component's specifications are updated as development progresses immediately taking experiences gained in account. This principle of the ability to update as work progresses is the reason to use a living document structure. GitBook is a well proven documentation tool to create, share and update living documents.

Versioning and tracking

GitHub and GitBook contain an elaborate versioning and tracking system known by the overall majority of European software developers, where the Track-changes and multiple simultaneous document update functions as available in MS Word are less known to them.

Share open-source code on GitHub

GitHub is a platform to share software code and work on this code by multiple developers at different locations. This sharing of code is required because:

- Different DRIVER+ partners from around Europe work on the Test-bed's development.
- All software created specifically for the Test-bed should be open source and thus sharable on a platform like GitHub.

Documentation and code together

Having the code and the specifications documents together on one online repository makes development work and sharing of this work much easier, findable and understandable for all the involved and for external parties. GitBook and GitHub originate from the same platform and are already linked to each other. Using GitBook next to GitHub is thus the most logical choice.

2. Work-process used to come up with the Test-bed's specifications

This chapter explains the process used to draft the functional requirement specifications, which are documented in this deliverable. This iterative requirements design-process was used by the Test-bed's core-development organizations (FRQ, TNO and XVR) and by the other development and review partners (AIT, ITTI and TCS; GMV, JRC and WWU).

The process followed is not an exact copy of a single, specific design or software-development method, but is based on the generic agile software development method, currently used by many software-developers worldwide.[2] This agile process encompasses multiple parallel design and development efforts going on at the same time. Experts of different domains (e.g. functional designer vs. software developers, or organization creating component A vs. organization developing component B), work on their own component and meet regularly to discuss the overall architecture and links between these components. The WP-leader (i.e. TNO) maintains an overview over all components and the high-over architecture of the entire Test-bed and thereby actively steers the requirements design process.

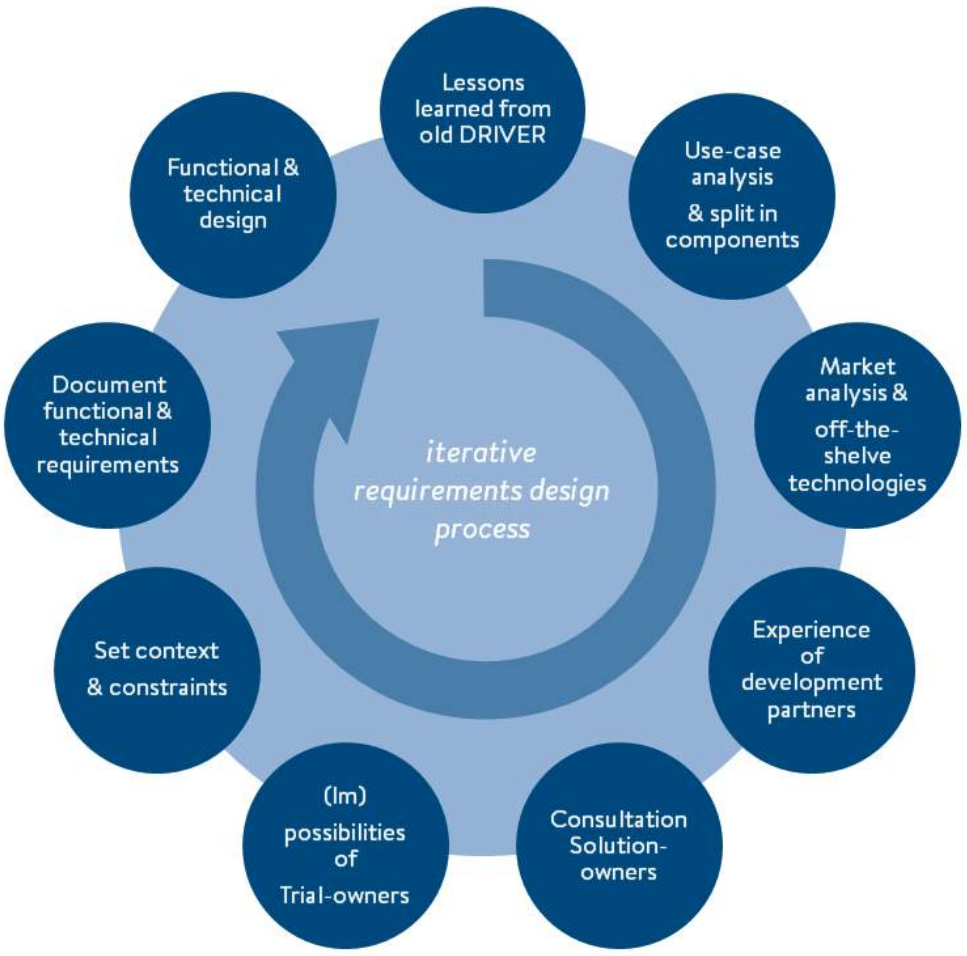
Lessons learned from old DRIVER
Use-case analysis & split in components
Market analysis & off-the-shelve technologies
Experience of development partners
Consultation Solution-owners
(Im) possibilities of Trial-owners
Set context & constraints
Document functional & technical requirements
Functional & technical design
iterative requirements design process

Figure 2.1: The iterative process to set the DRIVER+ Test-bed specifications

[2] en.wikipedia.org/wiki/Agile_software_development [2]

The iterative requirements design process followed is visualized in Figure 2.1. The following 9 steps can be distinguished in this process, but note that the order of these steps stated here differs in each round the iterative process is followed:

I. The design process is started by gathering **lessons learned from the previous phase of the project**. The experience of partners involved in Task 932.1 that has participated in former experiments have been taken into account. The outcomes are documented in the next chapter.

II. The **high-level use-cases** of the entire Test-bed are considered. This quickly led to the fact that the **Test-bed is split up in multiple components**, due to the different functionalities of these components and to be able to divide the work amongst multiple development partners.

III. For each component (and functional/technical sub-component) a **market research** is done about what is already available off-the-shelf, to prevent development efforts are wasted on designing and making everything from scratch while there is a sufficiently functioning component available. Sometimes these components can be directly included in the Test-bed (e.g. open source Kafka technology), sometimes parts are re-used from the previous phase of the project (e.g. partly the CIS architecture and base technologies) and sometimes only the functional design is reviewed (e.g. the NATO exercise management tool JEMM) and references are made to such products/technologies.

IV. Next to off-the-shelf components and technologies, also the **experiences of development partners** in similar projects are used to draft the Test-bed's specifications. For instance the Observer Module, as made by Middlesex University, Netherlands Aerospace Laboratory and XVR in the FP7-project CRISIS[3], served as a source of design input for the Test-bed's Observer Support Tool (see section 5.8).

V. Other **Solution-owners** within the DRIVER+ consortium are **consulted** regarding technical link-abilities, as one of the primary functions of the Test-bed is to enable solutions (in case these are software) digitally link up with the Test-bed to exchange information in between these solutions and with other components. Next to Solution-owners included in the DRIVER+ consortium, also potential external Solution-providers are considered, both because each Trial is open to externals and because of the required sustainability of the Test-bed.

VI. Similarly like consulting Solution-owners, the **possibilities and impossibilities at Trial-owners** are also reviewed. The Test-bed must be able to function at all Trial-owners facilities included in the DRIVER+ consortium. For the Test-bed's sustainability outside the DRIVER+ project it must also be relatively easy to implement the Test-bed at other locations.

VII. Steps IV, V and VI together define the **context and constraints** in which the Test-bed must be able to function. Next to the characteristics of the environment the Test-bed is implemented in (e.g. computer hardware can be located indoors and does not have to be packed weather-proof), this contextual description also provides a set of constraints on the Test-bed (e.g. Test-bed's core must be able to function without a high-end full-time available internet connection, as some Trial-owners cannot guarantee this availability 24/7 at their site).

VIII. All these requirements (i.e. in detail a **set of functional and technical requirements and wishes**) are documented. This documentation forms this deliverable D923.11. As know-how about the Test-bed's components and their functioning grows during development of these components and the use of them during Trials, D923.11 is a living document which is updated online whenever specifications are changed or elaborated.

[3] CRISIS-project coordinated by Middlesex University, which received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° FP7-242474 [16]

IX. The drafting of these requirement specifications is done in parallel with the **functional and technical design**. These design and the specification efforts highly influence each other. So although design decisions are documented in D923.21/D923.22/D923.23 (describing the Reference Implementation of the Test-bed), both these steps form an integral part of the requirements specification process.

This circular process of documenting the functional specifications is followed many times. When viewed from a distance, this process is followed during four main phases:

1. This process was followed already by the core DRIVER+ development partners, who also participated in the former phase of the project. Their experiences and ideas on improvements were agreed upon and documented in the WP923 description in the DOW. This resulted in the main concepts behind the new Test-bed.
2. At start of WP923, the core development partners TNO, XVR and FRQ had face-to-face and remote specification and design meetings, going through all of the Test-bed's components, what they are intended for, how they are linked and the high-over functional and technical design of them. The outcomes were documented in the meetings' minutes.
3. Following these high-level specification and design meetings, both internal meetings and face-to-face and remote meetings with other component developers were organised. The high-level specifications and designs were shared and these were then further detailed and documented in component specific documentation (see also chapter 5).
4. During the development of each component, the specifications and designs are further detailed and updated. This phase will continue when the Test-bed is used in Trials. The outcomes of this phase are the components and the updates on the living documents.

3. Lessons learned on the Test-bed from the former DRIVER-project

In the former phase of the project, several experiments were organised to test crisis management systems and solutions. This chapter will discuss the lessons that can be learned from these experiments, specifically related to the technical infrastructure (i.e. the Test-bed), encompassing the connection to the operational systems as well as the simulators.

An analysis of deliverables and internal documents[4] about former experiments learned that very little was documented about the experienced good and bad characteristics of infrastructures used within former experiments. So in order to collect the below stated lessons learned, several interviews were conducted with former experiment leaders and developers and architects involved in the former phase of the project.

3.1 Summary of the lessons learned

1. A generic Trial Guidance Methodology (TGM) is needed for analysing the need, and preparing, executing and reviewing an experiment.
2. Only one Test-bed must be created and this must be used for each and every Trial.
3. The Test-bed should be open source.
4. The Test-bed should have a message-oriented architecture.
5. The Test-bed should use well-defined, easily accessible, syntactically correct messages, and close to common standards.
6. The Test-bed should be clearly defined and scoped, i.e. what it is, and what it is not.
7. The Test-bed must be easily reproducible, and offer administrative as well as supporting tools and services.

All of these aspects are being addressed in DRIVER+. The experiences from the previous phase of the project, based upon which the lessons learned are identified, are described in the following sections.

3.2 The organization of each experiment was ad hoc

In the former phase of the project, there was no clear overall process to perform an experiment: how to prepare, execute and evaluate an experiment, which questions to ask, how to look for existing solutions, etc. Different partners had different ways of doing things, and there was no clear guidance or manual to steer them all. This lead to miscommunications, discussions, inefficiency, stress and less than optimal experiments.

Lesson learned: A generic Trial Guidance Methodology is needed for analysing the need, and preparing, executing and reviewing an experiment.

3.3 Each Trial created its own Test-bed

Each of the organised experiments in the first phase of the project created its own, unique, Test-bed to perform the experiment. Meaning the Test-beds used were based on for example:

- A legacy system owned by one of the consortium partners.
- The High-Level Architecture (HLA), an open but complex standard for connecting different simulators, but with only a few (expensive) commercial providers and a steep learning curve.

[4] Full list of experiments analysed for lessons learned is included in D922.21 - Trial guidance methodology and guidance tool specifications (version 1) [14]

- A semi-open source system based on Apache Kafka, a message streaming service.

This may have made sense for a small one-off project that does not need to be repeated, but for a pan-European project, a lot of resources were used inefficiently. In particular, this meant that:

- Every solution or simulator provider had to connect his system to different Test-beds, using different protocols.
- Every solution or simulator provider had to deal with different message types.
- Setting up and managing the IT infrastructure was difficult, and offered little learning opportunity as in practice the Test-bed use differed for each experiment.

Lesson learned: Each Trial must use the same Test-bed. Therefore, only one Test-bed must be created.

3.4 Test-beds were not open source

None of the used Test-beds were completely open source. Again, this does not matter within the scope of one project, were partners could use these Test-beds license for free, but if the Test-bed is to remain sustainable after the project, and be attractive to outside parties, this will be a serious roadblock. Besides obvious aspects as costs for getting a license, it would create a strong dependency on the owning party: each change someone likes to make would have to be approved by the owner. And after an approval, it would take time to implement the change request, potentially leading to serious delays.

In addition, from a developer's point of view, dealing with a large and complex closed source environment can seriously delay or even block his/her development efforts. Failures in the developer’s new component’s software may be caused by bugs in other linked, closed source components. Because these other component(s) are closed sourced, they cannot be fully examined by the developer directly, making it impossible to determine whether the issue is caused by his/her own software or by the linked component(s). This investigation then requires both the developer looking in his/her code and external developers looking in their code. And for any bug that is discovered in a linked closed source component, it is up to the owner of that component if and when to fix it. There is no chance to quickly find and fix it by the developer of the new component.

Lesson learned: The Test-bed should be fully open source.

3.5 All Test-beds used a message-oriented architecture

Each Trial needed to connect many, quite different, software systems: end-user solutions, incident simulators, 3D visualisations etc. But although each Test-bed was different, they all used a message-based infrastructure. The reasoning behind that is simple: when you have n systems, the number of connections between different systems will grow exponentially *(#connections = ∑ n-i, where i=1..n-1*). So it is better that the Test-bed uses a kind of mail service or message broker: if you want to communicate something, tell it to the broker and the broker will make sure that your message is delivered to the right person at the right time. So all messages go into one hub, and are then being distributed to all other applicable components. As a result all components only have to connect to this one hub, instead of separately to all other components. This components-hub architecture is in software development called a message-oriented architecture.

Lesson learned: The Test-bed should have a message-oriented architecture.

3.6 Each Test-bed used its own set of message standards

When different systems need to talk to each other, the messages they exchange must be well understood, at least at a syntactic level (*I can read your message*), and preferably also on a semantic level (*I understand your message*). The latter, however, has implications beyond what this project tries to solve here, as it would entail a universal translator that can translate any language in any other language including organization specific use of definitions. To further complicate matters, in a crisis management environment, a Test-bed has to deal with two types of systems: solutions and simulators, each with their own set of messages.

For simulators, often DIS (Distributed Interactive Simulation) or HLA (High Level Architecture) are used as standards to exchange messages. Syntactically, they can be easily interpreted, albeit often their strict schemas do not allow for sufficient freedom to express deviations from the standard, and certain fields may be misused to communicate this missing information.

For solutions in the operational crisis management domain, the CAP (Common Alerting Protocol), EMSI (Emergency Management Shared Information) and EDXL DE (Emergency Disaster eXchange Language) standards are often encountered. But also here, in practice different flavours can be seen throughout regions and organizations. Even a simple standard as CAP has different implementations across countries, but often also between different organizations. Furthermore, not all standards are freely available. For example one has to pay for the TSO/EMSI standard. And even if these costs are often negligible in a large project, it may cause delays, since a developer needs it, but has to get approval in order to buy it, and is probably not allowed to share it with project partners. For sustainability of the Test-bed, this is a disadvantage.

To deal with these issues, the Test-bed should have:

- A well-defined set of messages.
- Each message can be validated syntactically.
- A message should be close to the common standards (i.e. you are allowed to tweak the standard to suit your Trial).
- Each message's definition (often called schema) must be easily accessible to all participants.

Lesson learned: The Test-bed should use well-defined, easily accessible, syntactically correct messages, and close to common standards.

3.7 The definition of Test-bed was confusing

In the former phase of the project, the respective partners had no clear, consistent picture of the scope of the Test-bed. To perform an experiment, often most of the following components were needed:

- Operational systems: either legacy systems, or new systems that were to be evaluated.
- Simulators: to create a fictive incident, such as a flooding, earthquake or explosion, and a virtual representation of the world.
- Gateways for exchanging messages between operational systems, between simulators, and between operational systems and simulators.
- Support tools, for administrating the technical infrastructure, controlling an experiment and observing/reviewing the results.
- Location, i.e. the place where the experiment took place.
- Services, like sharing maps, census data, etc.

So for some people involved in experiments (e.g. Solution-providers), the Test-bed was only the gateway between operational systems. For others (e.g. the experiment organiser), it contained all of the above.

Lesson learned: The Test-bed should be clearly scoped, i.e. what it is, and what it is not.

3.8 Test-bed management was complex

The complexity is partially related to the previous observation that each experiment created its own Test-bed environment. However, the problem went deeper than this, since an experiment also needed:

- Administration of all systems: are all systems up-and-running; is everyone connected correctly, so they receive the expected messages; are some messages secured properly.
- Supporting tools: what is the main list of scenario events, and how to share this; what is needed to observe during an experiment, and how to collect these observations for an after action review; how to evaluate the outcome of an experiment.
- Data services: maps, map layers, weather reports, census data, etc. need to be shared between operational systems and simulators.
- Reproducing a Test-bed: In case an experiment is to be repeated.

Lesson learned: The Test-bed must be easily reproducible.

Lesson learned: The Test-bed must offer administrative as well as supporting tools and services.

4. Aims of the Test-bed in DRIVER+ and high-over functional design

This chapter first gives the overall aims of the Test-bed in relation to the objectives of DRIVER+ as a whole. It describes the environment in which the Test-bed should function and thereby the high-level functional design of the Test-bed. In section 4.4 the different types of users actively operating (i.e. using) the Test-bed are described and a high-level use-case for each use-mode is provided.

4.1 What the Test-bed should provide

In the DOW, the first stated objective of DRIVER+ is to *"Develop a pan-European Test-bed for Crisis Management capability development (…)"*. In more detail:

- This to-be-developed Test-bed should specifically provide *"(…) an infrastructure to create relevant environments, for enabling the Trialling of new solutions and to explore and share CM capabilities(…)",*
- because this Test-bed is used to *"Run Trials in order to assess the value of solutions addressing specific needs using guidance and infrastructure".*

So the entire Test-bed is supportive to assess solutions in a realistic, controlled and safe manner. Furthermore the Test-bed should not be a one-time implementation, but can be implemented within multiple organizations, both during the project, but also after the project and at organizations not being a project-partner. And the Test-bed will be delivered in 3 versions, to be able to take in the experience gained by using it in Trials and the Final Demonstration and have the ability to further develop it, both on the level of quality and on functionality.

Based on these DRIVER+ objectives and the lessons learned from the former phase of the project, the following high-level requirements are set. Note that this list consists of both real requirements (i.e. must do's) and wishes (i.e. should/could do).

1. **A realistic Platform to assess solutions**

   The Test-bed must be able to be implemented at a space - called Platform – in which Trials can be conducted in a realistic CM setting, in a controlled manner, in safety and not intervening with other (real CM) operations. This Platform is a physical and digital work-space for both the Participants (i.e. the CM practitioners using the solutions during the execution-phase of the Trial) and the Trial staff (i.e. all other people involved during the preparation-phase, execution-phase and evaluation-phase of the Trial).

2. **Test-bed is used fitting the Trial Guidance Methodology**

   The Trial Staff will conduct Trials using this Test-bed following the Trial Guidance Methodology and will receive support and training, such that each Trial is set-up, executed and evaluated optimally by the Trial Staff.

3. **Connect solutions with each other**

   The (mainly digital) infrastructure of the Test-bed must make it possible to connect the solutions in a Trial with each other, such that they can be controlled, can exchange data with each other and can be fed with simulated data

4. **Connect simulators to provide information of a fictive crisis**

   The infrastructure of the Test-bed must also provide one or more simulated incidents/crises, because trialling solutions during a real incident is both from an ethical and a practical perspective impossible. These simulated incidents - which can be completely virtual and computer-based, completely staged outdoors or a combination of the two - are to provide simulated data to the solutions via the Test-bed's infrastructure and are to provide information flows about the fictive incident(s) to the Participants (i.e. information flows not being presented via the solutions, for instance for incident commanders being present at a fictive incident scene an eye-level view on that scene).

5. **Interface to control the Trial**

   The Test-bed's infrastructure must provide an interface to the Trial Staff – specifically the Operators controlling the Test-bed's components – to create fictive incidents and to control how this fictive incident changes during the Trial's execution-phase and to control the availability of and the data to/from the solutions during this execution-phase.

6. **Interface to make observations**

   The Test-bed's infrastructure must provide functionality for the Trial Staff to save observations, recordings and data-logs coming both from the solutions and from the simulated incidents. During the evaluation-phase, these observations, recordings and data-logs are to be combined to support the creation of an evidence-based evaluation report of the Trial.

7. **Open source Test-bed**

   Crisis management affects us all, and therefore the Test-bed should achieve a broad adaptation and no hindrances with respect to license management. In addition, being open source, all developers from whatever organization are welcomed to create pull requests to improve the Test-bed's components.

8. **Open for commercial organizations**

   Commercial organizations can help to integrate the Test-bed in an organization, offer software solutions that exceed the capabilities of the originally offered components, or provide support in maintaining Test-bed components. The development and maintenance approach, while components are delivered in the project open source, should not exclude these commercial organizations.

9. **Standardized connections/interfaces**

   The Test-bed's infrastructure, specifically the software's data exchange interfaces, must be standardized as much as possible to ease up connection and joint use of many different solutions and simulators, both from within the project and external ones.

10. **Modularized Test-bed in several components**

    The Test-bed's infrastructure must be modularized and can be implemented partly and completely, such that it can be used efficiently and effectively during all development-, testing-, preparation-, execution- and evaluation-phases, by Trial-owners, Solution-providers, Simulator-providers and other Test-bed development parties.

11. **Provide data-sets for development and testing**

    Specifically for developers who link up solutions and simulators to the Test-bed and for Trial Staff involved in scenario creation, the Test-bed must provide functionality to easily obtain (simulated) data and component's inputs/outputs without the need that for every development step/test the entire Test-bed with all of its components must be activated, as this would require a too high overload on applications, computer hardware and manpower to perform even a simple test.

12. **The Test-bed shall run in a Docker environment**

    Installing software can be tedious, and due to its nature, the Test-bed consists of an integration of many tools. And although each tool can be installed separately, in order to facilitate an easy deployment, Docker images and Docker compose will be used to rollout the Test-bed. Docker images are ready-to-use installer applications to install a specific software component on a computer. Docker compose provides a process to combine multiple Docker images together, such that all can be installed within one process, resulting in little manpower needed to implement any combination of Test-bed components. As Docker is supported on Unix as well as Windows operating systems, many different organizations should be able to run it smoothly.

    As a consequence, for the moment most of the supplied tools will run in a Unix environment, since Windows-based Docker containers are still experimental. However, this will not limit developers to create Windows-only solutions, as it can still connect to the Test-bed.

13. **The core of the Test-bed is based on Apache Kafka**
    In order to integrate different tools within the Test-bed, and to make it easy to connect them with other solutions, Apache Kafka is selected as base integration platform. And Kafka will be the only integration platform used. Although there are many good open-source messaging systems available, Kafka has a number of distinct advantages:

    - It is supported by the Apache organization, and used actively by a number of major commercial organizations. This means that it will not go away quickly, and it has many client connectors to ease integration.
    - It is simple to setup and run, and performs very well. Where similar software systems cannot deal with more than 10.000 messages per second, Kafka can process a tenfold.

    So while there are other environments that could satisfy the DRIVER+ needs, this current reference implementation relies on Kafka, because the Test-bed should be a good reference implementation leveraging the efforts of the Kafka community. Being more generic by supporting multiple back-ends would require a lot more development work, as well as not being able to leverage the particular strengths of the backend.

    Although this actually is a design decision supported by all development organizations, during further design meetings it turned out that this decision has so much impact on the design of all components; therefore, this decision is stated here. By taking this decision, it forms a technical requirement on all components.

14. **Linked to the Training Module**

    And finally, the Trial Guidance Methodology (TGM), the tools to use this methodology and the Test-bed must be accompanied by a Training Module, primarily aimed at people organizing Trials (i.e. the Trial Staff) explaining how to use the TGM and all of the Test-bed's components best.

4.2 Positioning of the Test-bed

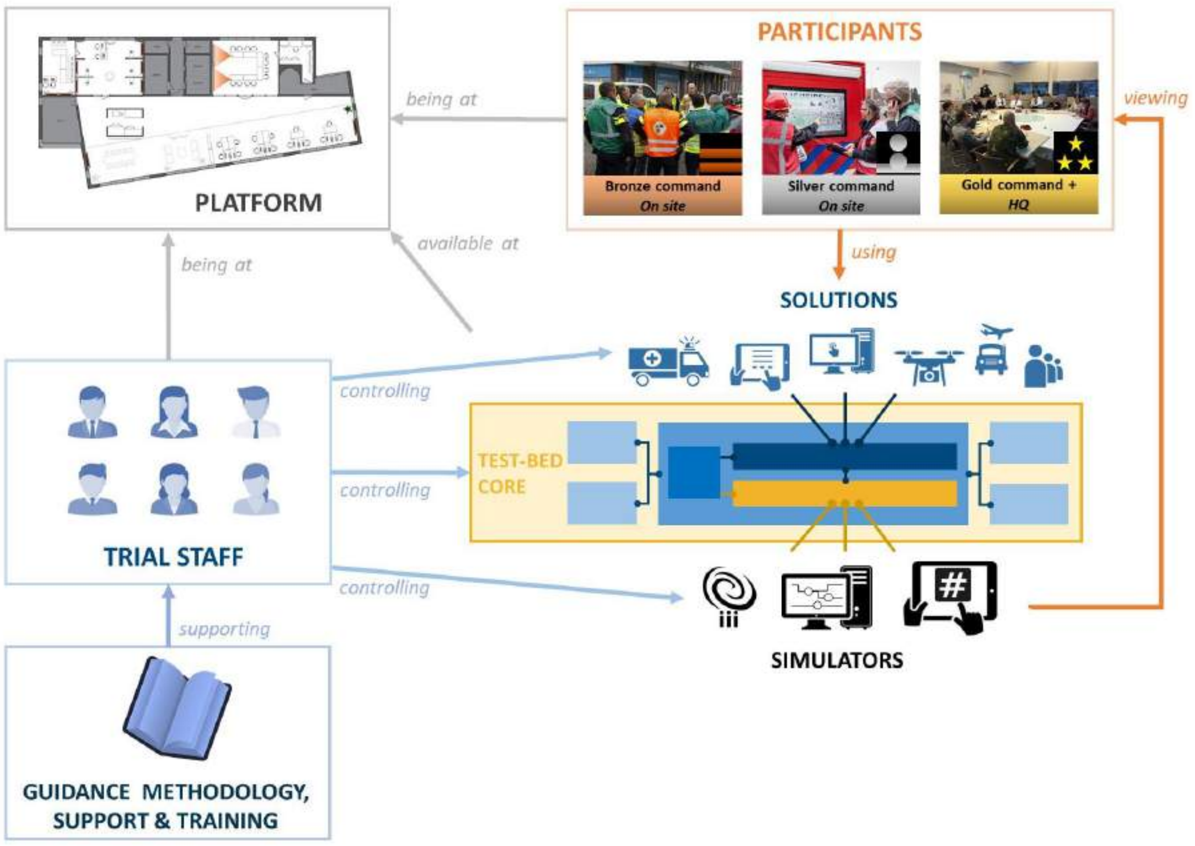

Figure 4.1: The Test-bed in its environment

The Test-bed is not just a stand-alone collection of software applications. It is used within a larger environment of DRIVER+ entities, as visualized in Figure 4.1.

- **Participants** are using the **Solutions** to solve a fictive crisis. Solutions are any type of tool/product (e.g. a software package, a training method or a new standard operating procedure) that is intended to address a practitioner gap and to support/improve Crisis Management.
- This fictive crisis exists in a virtual world and is thus a "virtual crisis". This virtual crisis is created by a set of simulators (this can be computer-based **Simulators** to simulate a crisis within a virtual reality world, but a fire brigade's physical exercise terrain is also a type of simulator to stage a physical yet fictive incident, and it is also possible to use a combination of both).
- Both solutions and simulators are connected to the **Test-bed**. The Test-bed's core makes it possible to connect the solutions with each-other and feed them with data coming from, or flowing to the simulators. The functionalities of each component of the Test-bed are explained in the next paragraph.
- The **Trial Staff** uses the Test-bed during the:
  - *Preparation-phase* – to build the scenario, set-up and test all solutions to be trialled, and set-up and test the simulators to support this.
  - *Execution-phase* – to control how the virtual crisis scenario evolves, observe the Participants and gather data-logs from both the solutions and the virtual crisis present in the simulators.

  - *Evaluation-phase* – to gather and analyse all observations, recordings and data-logs, and base an evidence-based evaluation report on these.

- The Trial Staff is supported - during all phases, but particularly in the preparation-phase - by the **Trial Guidance Methodology (TGM)**. The **Guidance Tool** supports the Trial Staff by leading them step-by-step through all phases of the Trial, thereby assuring that the Trial Guidance Methodology is followed correctly and lessons learned are used most effectively. This together with practical and logistical information is documented in the Trial Action Plan (i.e. an elaborate document about how the Trial is set-up, how it is to be run and what to evaluate). The **Training** Module provides the Trial Staff with training on this TGM, on using the Guidance Tool and Trial Action Plan and on how to implement and use all components of the Test-bed best.
- The **Platform** provides the physical space (i.e. one or more locations) and hardware to run the Trial (e.g. a simulation centre like available in The Hague, but this can also be a live exercise terrain). So the Test-bed, solutions and simulators are available there, it provides rooms/areas for Participants to execute CM operations with the solutions as they would do in real-life crises and provides rooms/areas for Trial Staff to operate the Test-bed and simulators, thereby controlling how the Trial runs. The Platform is also used in the preparation phase, by installing and testing all (software) components needed during the running of the Trial and for evaluation of the Trial.

4.3 High-level functional design

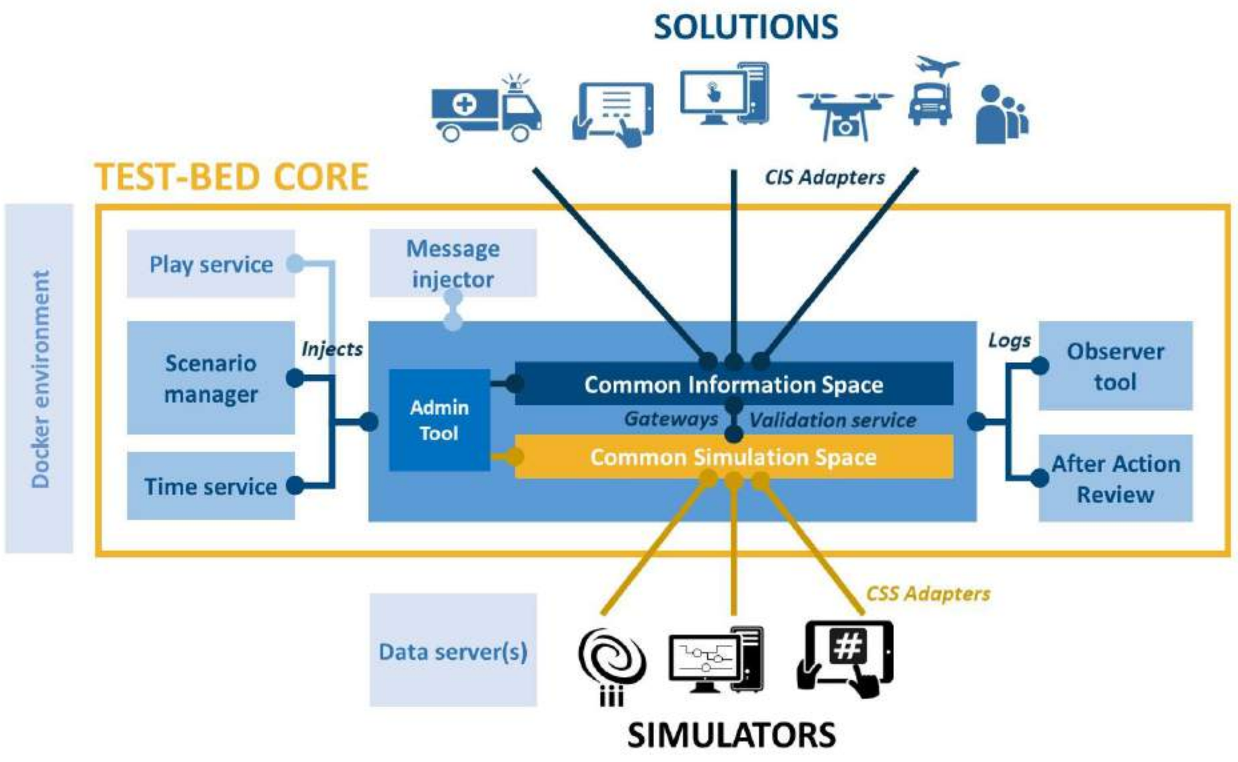

Figure 4.2: All components of the Test-bed

Zooming in on the Test-bed, as shown in Figure 4.2, the following components can be distinguished:

- The **Common Information Space (CIS)**, located in the central software connection bus, provides an interface to connect the solutions to the Test-bed. These connections are made by using the standardized CIS Adapters.
- The **Common Simulation Space (CSS)**, provides a similar connection bus to connect the simulators to the Test-bed. These connections are made by using the standardized CSS Adapters.
- Because the type of data-connections and the required data-exchange speed differs for solutions on one side (i.e. report-like messages with a lower update speed) and simulators on the other side (i.e. item oriented data with high update speed), the CIS and CSS are split. They are connected via one or more **Gateways**. Gateways are used from CSS to CIS to feed solutions with data from the simulated crisis (e.g. positions of virtual police cars driving around in the fictive region) and from CIS to CSS to pass through changes inputted via one or more solutions to the simulators (e.g. a command police to block off an area, resulting in several police cars driving to the edges of that area and setting up road blocks there).
- A **Validation Service** can be added to safeguard messages sent between CSS and CIS are correctly filled and understandable by the counterpart.
- Both the CIS and the CSS are controlled via the **Test-bed Admin Tool**. This provides an interface for Trial Staff to control which solutions and simulators are to be started and which data is (not) exchanged.
- The **Trial Scenario Manager** and the **Time Service** components provide functionality and an interface for Trial Staff to:
    - In the preparation-phase: create a high-over timeline of the Trial Scenario with Injects of changes to the virtual crisis which can be activated during the execution-phase (note that it is far from necessary that all injects are activated and that this time-line in most cases has a branched structure of all things foreseen that could happen during managing the virtual crisis).
    - In the execution-phase: activate the needed Injects based on decisions (not) taken by the Participants, the use by them of the solutions and/or specific detailed storylines to be activated to trigger specific use-cases of one or more solutions. These Injects then trigger prepared changes in the Test-bed manager (e.g. simulate failure of internet leading to solutions not receiving data from simulators anymore), in the CIS or solutions directly (e.g. new message of role-played stakeholder) and mostly in one or more simulators (e.g. wildfire expands on the West flank).
    - In the execution-phase: influence the fictive time clock, meaning starting the Trial (i.e. play the time from a fictive point in time, for example a fictive climate summit in October 2020), influencing the speed of time of the fictive clock (e.g. speeding up time to go through CM operations quicker than in reality or slowing it down to focus on specific moments during CM operations), pausing the fictive clock (e.g. for a coffee break or a time-out for extra explanations), jumping back in fictive time (e.g. re-doing a specific CM operation) and stopping the fictive time (i.e. ending the Trial)

- The **Observer Support Tool** provides a module to prepare, make and gather observations:
    - In the preparation-phase: the Observation Manager (i.e. person within the Trial Staff) can manage who can observe who and create observation templates dedicated on what is to be observed, based on the data collection plan and set-up according TGM guidelines. Ethical considerations regarding the formulation of observation templates and storage of data are to be taken into account during this phase.
    - In the execution-phase: Observers (i.e. one or multiple persons within the Trial Staff) can create observations based on prepared observation templates.
    - In the execution-phase: all observations are gathered and the Observer Manager sees them incoming and can guide the Observers by extra Observation Templates and sending them messages.
- The **After Action Review (AAR)** module gathers in execution-phase the data-logs and (screen-) recordings coming via the CIS from the solutions and via the CSS from the simulators and gathers the Observations coming from the Observation Support Tool. In the evaluation-phase these data-logs, recordings and observations can be reviewed.
- For development and implementation purposes, the Test-bed also contains:
    - A **Message Injector** to input specific, single messages during development to test whether these messages are correctly received by another component (e.g. a fire alert message which is to be produced by a simulator is correctly received by a solution).
    - A **Play Service** to input several messages in chronological order, to test that without the Trial Scenario Manager and simulators fully up and running these messages are correctly received by the solutions or vice versa.
    - **Data Server(s)** to feed the simulators and potentially also the solutions with geo-specific information (e.g. the region The Hague, all of its fire stations and all fire truck usually present there; or a full digital map with GIS-data of Vienna)
    - A **Docker Environment** providing images (e.g. installers) of each component, such that the installation of a limited Test-bed configuration at a Solution-owner and installation of the entire Test-bed at a Trial owner’s simulation centre can be done quickly and transparently.

The next chapter (chapter 5) provides summarized overviews per component, including a hyperlink to that component’s separate detailed specifications document.

4.4 User-types and high-level use-cases

The following high-level use-cases describe the workings of the complete Test-bed during the different phases of use. Note that developing and updating the Test-bed’s components is left out of these phases of use. The use-cases are focused on how users operate components of the Test-bed. This section is intended to describe the coherent use of the complete Test-bed and to further explain the functional design of the Test-bed. This section is not intended to detail each specific component. Therefore, the steps in the use-cases are described only on a high-over level.

4.4.1 Use-modes

The Test-bed is used throughout 3 phases and next to that in Development-mode (i.e. use-modes of the Test-bed - Table 4.1).

Table 4.1: Use-modes of the Test-bed

Use-mode	Description
Development mode	In which the Test-bed's components are developed or a solution or simulator is connected to the Test-bed.
Preparation phase	In which all solutions and simulators are connected to the Test-bed, the Scenario is built and the entire set-up is tested to assure a good run of the Trial.
Execution phase	In which one or more sessions are run with Participants using the solutions according to the evolving Scenario.
Evaluation phase	In which the Trial's sessions are analysed and conclusions are drawn about the usefulness of the solutions and the way the Trial was executed.

4.4.2 User-types

Throughout all use-modes, the user-types as listed in Table 4.2 can be distinguished. Note that one member of the organizing Trial Committee can take up the roles of several user-types (e.g. one person being a Trial Scenario Builder and the Trial Director and one of the Trial Evaluators). Also note that the entire set of roles to be included in the preparation and execution of a Trial (as listed in the Trial Action Plan template) is longer than the list provided in this paragraph. This paragraph focusses the user-types only on those roles actively using one or more components of the Test-bed.

Table 4.2: User-types of the Test-bed

User-type	Description
Participants	Persons (usually CM practitioners) actively using the solutions and performing CM-operations based on which information they receive from the solutions and directly from the simulators about the fictive crisis.
Trial Scenario Builders	Persons from the Trial Staff who are configuring the Trial Scenario in the Test-bed's components (i.e. Trial Scenario Manager, simulators, Observation Support Tool, Test-bed Admin Tool and potentially also by configuring solutions).
Trial Director	The single leader of the Trial Staff (i.e. 1 person) being, in the execution phase, in charge of how the Trial and its Scenario evolves during the session(s).
Operators	Persons from the Trial Staff who are operating the Test-bed's components during the execution of the session(s), with a primary focus on the Trial Scenario Manager (i.e. how the Scenario evolves), the Test-bed Admin Tool (i.e. whether everything is working correctly and controlling information flow to/from solutions and simulators) and directly in the simulators (i.e. how the Scenario evolves).
Role-players	Persons from the Trial Staff who are supporting the Operators by providing role-play towards the Participants, thereby simulating stakeholders not present (e.g. performing phone-calls as minister or King of the Netherlands).

User-type	Description
Observer Manager	Person of the Trial Staff (i.e. usually 1 person, but can be multiple) to operate the Observer Support Tool both in the preparation phase (configuring the tool for a specific Trial) and in the execution phase (i.e. managing the Observers).
Observers	Persons who take observations during the Trial's session(s) by using, most likely, the Observation Support Tool. Observers belong to the Trial Staff, independent whether they are employed by DRIVER+ internal organizations or externals.
Trial Evaluators	Persons from the Trial Staff who analyse the Trial's session(s) during the evaluation phase, using the After Action Review component.
Implementers	Technical support staff who install and test one or more of the Test-bed's components at a Platform location.
Solution-owner	An organization that has developed a solution which is to be trialled using the Test-bed. For this, technical developers of that organization have to link their (IT) solution with the Test-bed via a CIS Adapter, thereby using (a sub-set of) the Test-bed's components, including the extra developer components.
Simulator-owner	An organization that has developed a simulator which is used during a Trial to provide information from a fictive crisis. For this, technical developers of that organization have to link their simulator with the Test-bed via a CSS Adapter, thereby using (not all) Test-bed's components, including the extra developer components.
Test-bed Component Developers	Persons working within the organizations creating the Test-bed's components. These developers also use other components of the Test-bed to develop and test their own component.

4.4.3 High-level use-case: Developing the Test-bed's components

This use-case is executed in Development mode, upfront of (i.e. initial development) and parallel to (i.e. component updates) the preparation, execution and evaluation of the Trials.

1. Developers draw out a functional and technical design of a component and the test-cases, taking into account the specifications of the other components their own component is linked to.
2. Developers develop the component and, if needed, adjust the functional or technical or test-case design.
3. If needed, developers contact developers of other components to adjust the specifications.
4. Developers test their own component stand-alone and in combination of the other components it is linked to.
5. Developers create a Docker image of their component and upload it together with documentation to the Docker Environment.

4.4.4 High-level use-case: Developing the connection of a solution/simulator

This use-case is executed in Development mode, upfront of the execution and evaluation of the Trials, and for bug-fixing during the preparation phase of a Trial.

1. Solution/Simulator-owner is being trained/instructed about the aims of the Test-bed, the available CIS/CSS Adapters and how to use and not use them.
2. Solution/Simulator-owner selects the most applicable CIS/CSS Adapter and develops this into a Solution/Simulator specific adapter.

3. Solution/Simulator-owner implements the Test-bed components needed to test its own Solution/Simulator and the specific adapter created
4. Solution/Simulator-owner tests its developed Solution/Simulator and the specific adapter and adjusts it if needed.

4.4.5 High-level use-case: Set-up and testing of the Test-bed

This use-case is executed at every location where the Test-bed is (partly) installed or updated, so before the Trial's execution phase starts.

1. Implementer, using the Docker Environment, creates a full installer of all components to install.
2. Implementer installs this installer on the hardware available at the Platform location.
3. Implementer also sets-up/installs the solutions and simulators needed for a Trial.
4. Implementer tests whether the installation was successful.
5. If issues come to light during these tests, the Implementer contacts the applicable Component Developers, Solution-owners and/or Simulator-owners.

These steps are repeated until the installation of all components, solutions and simulators is successful, such that the Trial can be prepared and run.

4.4.6 High-level use-case: Starting-up the Test-bed

This use-case is executed at the start of every use-mode of the Test-bed.

1. Operator or Implementer starts up the Test-bed Admin Tool, the Trial Scenario Manager and the solution(s) and simulator(s) that require a manual start-up.
2. Operator or Implementer selects the applicable Scenario, which starts up the other required components.
3. Operator or Implementer checks correctness of start-up of all needed components via the Test-bed Admin Tool.
4. Operator or Implementer restarts components and/or reloads Scenario if some start-ups were unsuccessful.

4.4.7 High-level use-case: Scenario creation

This use-case is executed in the preparation phase of each Trial.

1. Scenario Builders follow the Trial Guidance Methodology using the Guidance Tool to come up with a detailed Trial Action Plan (TAP) about what to trial, how to trial and what the Scenario will be.
2. Scenario Builders configures the components and selected solution(s) and simulator(s) according to the TAP.
3. Scenario Builders select data set(s) from the Data Server(s) to populate the solution(s) and simulator(s).
4. Scenario Builders enter the Scenario's timeline(s) in the Trial Scenario Manager.
5. Scenario Builders set a fictive start-time in the Time Service.
6. Scenario Builders set up a fictive, virtual crisis inside the simulator(s) and prepare the injects in these simulators that could occur.
7. Scenario Builders configure the possible data exchanges between CIS and CSS by setting up the Gateways and the Test-bed Admin Tool.
8. The Observation Manager configures observer roles and observation templates in the Observer Support Tool.
9. Scenario Builders, Observer Manager and future Operators and Trial Director run one or more test-sessions to check the entire Scenario and make adjustments if needed.

The last step is repeated until all involved feel comfortable with the set-up and Scenario to run successful Trial sessions.

4.4.8 High-level use-case: Training (extra) Trial Staff and Participants

This use-case is executed just before the execution phase of each Trial.

1. Scenario Builders and the Trial Director (and potentially Simulator-owners) instruct Operators new to the Trial and its Scenario about the Trial Scenario, what can be adjusted during the Trial execution, under which conditions and how.
2. Scenario Builders, the Trial Director and Operators instruct Operators new to the Trial about the Trial Scenario and which injects can be role-played and how.
3. Observation Manager instructs the Observers about the Trial Scenario and what is to be observed by whom and how to use the Observer Support Tool and the prepared observation templates.
4. Scenario Builders and/or Solution-owners instruct Participants about how to use the Solutions.
5. Scenario Builders and/or Simulator-owners instruct Participants about how to use the applicable simulator interfaces.
6. Trial Director gives a briefing to all Participants and Trial Staff about the upcoming session.

4.4.9 High-level use-case: Running a Trial session

This use-case is the execution phase of each Trial, focussed on a single session (i.e. 1 time a Scenario is run).

1. All Participants and Trial Staff take position at their respective locations.
2. Trial Director, with help of Operators and Observers, checks readiness of all Participants and Trial Staff.
3. Trial Director gives the ‘go’ to start session.
4. Operator starts the running of the fictive time in the Time Service and thereby the evolution of the Scenario in the Trial Scenario Manager.
5. Participants acquire information about the fictive crisis via the solutions and simulators.
6. Participants use the solutions as much as possible as they would do during a real crisis and take decision accordingly.
7. Observers gather observations following the observation templates in the Observer Support Tool and the Observer Managers sees these coming in and directs the Observers.
8. Operators and Role-players activate injects based on the decisions taken by Participants, the Scenario time-line(s), cases to trial the solutions on, observations received and over-all directions by the Trial Director.
9. When all applicable Scenario time-lines and cases to trial the solutions on are activated, or the Trial session has diverted too much from the Scenario, the Trial Director decides to stop the session.
10. The Operator stops the Scenario and fictive time progression via the Trial Scenario Manager and Time Service.
11. The Operators use the After Action Review component to gather and save relevant data-logs, recordings and observations of that session. Data not selected for later evaluation is deleted.
12. Observers or Evaluators might question the Participants, if the evaluation process states so.

In case of an error or crash during a session, these steps can be repeated to start a new session from the point before the error/rash occurred.

These steps can also be repeated from a point half-way the same Scenario to trial the solutions under the conditions if another decision was taken and thus other Scenario time-lines were followed.

One can also load another Scenario and repeat the steps above, thereby creating a new session and trialling the solutions under completely other conditions.

4.4.10 High-level use-case: Evaluating a Trial

This use-case is the evaluation phase of each Trial.

1. Evaluators view the data-logs, recordings and observations from a single session, using the After Action review component.
2. Evaluators analyse the combination of data-logs, recordings and observations from one or multiple sessions to research how exactly the solutions were used, how these solutions performed under certain circumstances and how the Participants reacted under these circumstances.
3. Evaluators can contact Solution-owners or even Simulator-owners to extract more information from specific data-logs and recordings.
4. Evaluators can contact Participants to gather more information about specific moments during a session.
5. Evaluators draw conclusions about these analyses into the usefulness of the solutions, the way the Trial was set-up and executed and refer back to the Trial's objectives. All this is documented in the Trial's evaluation report.
6. Data analysed or not needed anymore is deleted.

5. Summarized description per Test-bed component

This chapter contains for each component an overview of what it is aimed at, who uses it and in which phase of the Trial it is used, in the form of a table per component. If applicable, the tables also contain a hyperlink to the detailed specification of this component.

5.1 Common Information Space (CIS)

Table 5.1: Common Information Space

Item	Description
Short description	The Common Information Space (CIS) describes the concept of information exchange between tools that are connected to the Test-bed. Also, information provided by simulations are forwarded and distributed to the tools via the CIS, and vice versa, messages sent from the CIS are transmitted to the CSS.
Who will use it	Directly: - Developers Indirectly - Everyone using a solution connected to the Test-bed
Main functions	The CIS will account for: - Connecting the tools to the Test-bed via CIS adapters - Distribution of messages - Security regarding access rights (They have not been defined)
Functions it does not do	The CIS will NOT account for: - Validation of messages
Links with other components	- Admin tool – to configure (parts of) the CIS - AAR – re-uses messages sent via the CIS - CIS-CSS Gateways – data exchange between CIS and CSS in both directions - Validation Service – messages coming from the CSS to the CIS are syntax validated
(Technical) conditions	Apache Kafka is needed
Reference to repository/details	

5.1.1 Extra notes

The CIS itself is visible only to developers, not end-users. However, end-users may configure the CIS (or parts of it) to a certain extend using the Admin tool described in paragraph 5.5.

5.2 Common Simulation Space (CSS)

Table 5.2: Common Simulation Space

Item	Description
Short description	The Common Simulation Space (CSS) describes the concept of information exchange between simulators that are connected to the Test-bed. These simulators jointly generate and maintain a simulation world needed for providing a ground truth of the fictive crisis, feed the solutions with fictive data via the CIS – and vice versa – receive messages sent from the CIS, and to deliver to the Participants a good enough image of the fictive crisis for them to be assessed.
Who will use it	Implementers, Solution-owner, Simulator-owner, Test-bed Component Developers
Main functions	The CSS will connect the simulators to each other and link to the gateways to exchange data CIS-CSS (bidirectional)
Functions it does not do	Deal with security: Simulators don't have secrets. Semantic validation/translation of messages send over the CSS. Other message format support (like JSON or XML).
Links with other components	Individual simulators (bidirectional, but note these are no Test-bed components); Admin tool for configuration of the CSS; CIS-CSS gateways; Trial Scenario Manager (bidirectional); Time Service (from Time Service to CSS); AAR to use the data for analysis (from CSS to After Action Review)
(Technical) conditions	Open source; Standardized connectors via the CSS Adapters; Possible to run in a Docker environment; Based on Apache Kafka
Reference to repository/details	

5.2.1 Extra notes

Simulators all have their own data model of how they represent the simulated world. The CSS allows these simulators to agree on a communication form that the simulators understand to create and maintain a joint simulated world. Next to the CSS, there also is the Common Information Space (CIS), that is used to connect all the solutions with the Test-bed and thus with each other. The design decision to not connect the simulators to the CIS directly is mainly to ensure the two spaces of simulated truth and perceived/ communicated truth are kept separate inside the Test-bed.

Like with a lot of emergency management processes, obtaining relevant information from the real world to base a decision on is either done by actually being at a specific location observing the current state, or receiving and sending messages via all kinds of communication channels from persons or systems at a specific location (e.g. radio communication, sensor input, camera feeds). These ways of obtaining information gives a (shared) perceived truth to be used in further emergency management decision making. However, due to a wrong observation or miscommunication, the perceived truth can be different than the simulated truth. The simulators should only be concerned with maintaining the current state of the simulated truth (including entities and processes), and shouldn't have to deal with the different kinds of communication types for solutions and users to create the perceived/communicated truth.

The Common Simulation Space allows simulators to only focus on maintaining the current state of the simulated world (i.e. the simulated truth of the incident and the world around it). In order to communicate state changes with other simulators inside the CSS, self-created communication messages are allowed inside this space. This is different than the messages being sent over the CIS, because the CIS is more aligned with current emergency management standards (like Common Alerting Protocol (CAP) messages, or Emergency Data Exchange Language (EDXL) messages). In order to direct the simulated world towards a desired scenario relevant for the Trial, the Trail Scenario manager should be able to send out messages to change the simulated world. This should be done via inject messages that all simulators can understand in order to execute the requested inject (e.g. start the breach, let the container explode).

5.3 CIS-CSS gateways

Table 5.3: CIS-CSS Gateways

Item	Description
Short description	The CSS-CIS gateways are the interface between the Common Simulations Space (CSS) and the Common Information Space (CIS). Data from the simulations is translated into data that can be understood by the tools connected to the CIS and vice-versa. Since they translate specific message types, there may be many of them.
Who will use it	Developers
Main functions	The CSS-CIS gateways will account for: - Aggregating messages coming from the CSS - Translating between CSS topics and CIS topics
Functions it does not do	The CSS-CIS gateways will NOT account for: Semantic translation
Links with other components	- CIS & CSS: the gateways are the bridge between them Validation service: the validation service is a gateway service that validates messages going through the gateway
(Technical) conditions	Must run in a Docker environment
Reference to repository/details	

5.4 Validation Service

Table 5.4: Validation Service

Item	Description
Short description	The message Validation Service serves the purpose of validating the syntax of messages to match a certain standard. Only successfully validated messages are distributed further inside the system. It is an extra step with respect to the default validation that is carried out by all adapters, and mainly required when testing new solutions or during a dry-run.
Who will use it	It is internally used by the adapters, and configured via the Admin tool
Main functions	The validation service will account for: - Checking if the syntax of the message fulfils a certain standard - Listening to a specific TOPIC (e.g. validation-cap-topic). if it validates, the messages will be forwarded to the corresponding standard topic (e.g. CAP-topic).
Functions it does not do	The validation service will NOT account for: Semantic validation of messages
Links with other components	- CIS-CSS gateways: the message validation is a gateway service CIS: only messages that fulfil a certain standard are forwarded to the rest of the system
(Technical) conditions	Use of Apache AVRO schemas to represent standards
Reference to repository/details	

5.5 Test-bed manager (Admin tool)

Table 5.5: Test-bed manager (Admin tool)

Item	Description
Short description	The Admin tool can be used to configure parts of the Test-bed, e.g. which tools are connected to each other (via the topics). Also, it provides Test-bed information (i.e. existing topics) as well as status information (heartbeat, configuration, logging) of the connected tools. Furthermore, it is the main interface to the security services.
Who will use it	Trial manager
Main functions	The Admin tool will account for: - An interface to configure the CSS-CIS gateways - An interface to configure the adapters - Loading the KAFKA topic configurations - Providing a Watch dog

Item	Description
Functions it does not do	The Admin tool will NOT account for: - Setting up topics after initialization - Modifying topics after initialization
Links with other components	- CIS-CSS gateways and adapters: they are configured using the Admin tool
(Technical) conditions	Must run in a Docker environment
Reference to repository/details	github.com/DRIVER-EU/test-bed-admin [3]

5.6 Scenario Manager

Table 5.6: Scenario Manager

Item	Description
Short description	The Scenario Manager can be used to create scenarios (master event lists) that are injected, via the Test-bed, into the CSS or CIS. For example, it injects a message to start a flooding, to send out emails to participants, or to instruct a role-player to perform an act. Naturally, it can also control the time, and (re-)start/pause/stop a scenario.
Who will use it	Trial manager, scenario writers
Main functions	Publish messages, often intended for simulators, but can also be used to directly show a message inside a solution.
Functions it does not do	Record messages
Links with other components	Can be used to send messages to simulators, but also to solutions, via the adapter. Its strongest connection is with the Time Service in order to control the fictive scenario time.
(Technical) conditions	Must run in a Docker environment
Reference to repository/details	github.com/DRIVER-EU/scenario-editor [4]

5.7 Time service

Table 5.7: Time service

Item	Description
Short description	The Time service is the single source of truth of the fictive time during a Trial. It listens to the scenario manager in order to play/pause/stop a scenario, and it may speed up or slow down the simulation. Each adapter will subscribe to the time service, and offer its users the fictive time. When sending messages containing time information, each service should use this fictive time. In addition, it shares the server's time using the Network Time Protocol (NTP), so all services in the Test-bed can use it to sync their clocks.

Item	Description
Who will use it	It is an internal service
Main functions	Publish the fictive scenario time, play/pause/stop/speed up/slow down the time
Functions it does not do	Run the simulation time backwards or time-jumps, as this would corrupt simulation processes. (Note that reloading a previously saved moment during a Trial is possible and is a way to do a time-jump backwards from an end-user's perspective.)
Links with other components	Scenario manager and all adapters
(Technical) conditions	Must run in a Docker environment
Reference to repository/details	github.com/DRIVER-EU/test-bed-time-service [5]

5.8 Observer Support Tool

Table 5.8: Observer Support Tool (OST)

Item	Description
Short description	The aim of OST is to collect observations, inform observers about Trial progress, and visualize collected data. The ethics regarding observation collection as decided upon in the Trial design must be taken into account by the user creating observation templates (i.e. what to collect how).
Who will use it	Trial Manager (person who configures the OST upfront of a Trial, manages Observers and has overall control over the system). User, which can be an Observer or Participant
Main functions	Allowing Trial managers to setup observation questions and send messages to observers Assigning observers to specific observation tasks Allowing observers to enter observations Allowing Trial managers to monitor observations, also in real–time Allowing participants to complete surveys
Functions it does not do	OST does not provide sets of data about events and time. This tool waits for simulation phases from external system, publish events and generate questions based on these data.
Links with other components	OST is linked with Test-bed, which is responsible for providing simulation phases – data about events and simulation time. Package of data is sent to OST Server, events and set of questions are generated. When the new event is displayed, OST Server notifies the user and set of questions changes.
(Technical) conditions	Observer Support Tool (Mobile): - Web browser - Android (tablet and smartphone)

Item	Description
	- iOS (tablet and smartphone) Observer Management Tool (Desktop): - Web browser
Reference to repository/details	github.com/DRIVER-EU/ost [6] driver-eu.gitbooks.io/specification-of-the-online-observer-support-tool [7]

5.8.1 Extra notes

The Observer Support Tool's aim is to collect observations, inform observers about Trial progress and visualize collected data. There are different perspectives to look at this tool. The main user, who uses the mobile version of a tool to send his observations, is called Observer. From the other side there is a Trial Manager who focuses on collected data and analysing it on desktop. Each of them has their own functionalities provided by OST. There are also these functionalities which are connected with non-functional requirements.

Observer Support Tool provides different views for each user. Observer sees name and description of a Trial, events that have already happened and observation templates which Observer can fill in, whereas Trial Manager has displayed summary of all observations that have been sent; in addition, the Trial Manager can even see a summary of observations in time and messages the Trial Manager sent to Observers. The Trial Manager is responsible for assigning role to the user which can be an Observer or Participant of a Trial.

First, the Observer selects the Trial of interest. The Observer sees its name, description and list with events that have already happened. The Observer can send an Observation by answering questions that are connected with events – each new event is a trigger and can change the set of questions.

OST Server does not provide data about events; it is responsible for data exchange but not for preparing them. It receives data packages about events and simulation time from Test–bed and reacts on triggers. Events can be also sent directly to users by Trial Manager.

If events are sent from Trial Manager, OST Server publishes them both to Test-bed and to the user. The Trial Manager not only manages the Trial but also prepares the environment to obtain information that is needed. The Trial Manager is responsible for projecting questions and question types. Contents and number of questions are optional, only the form is imposed.

When Test-bed sends package with data about events and time, OST Server notifies Trial Manager and users about new event. OST Server reacts on triggers and matches proper set of questions, which are sent to Trial Manager and then published to Observer. When Observer sends his Observations, Trial Manager collects obtained data and has displayed reports about them and, if needed, can generate it in CSV (i.e. comma separated values file-format).

Questions have different purposes, which are indicated in Observation Types. The big advantage is getting better criteria of comparison. Database with observations is more varied but it also connects categories, so analysing is more efficient. Correctly prepared questions and labels lead to more efficient results and better conclusions. Questions can also have different answer types such as slider, checkboxes, radio buttons and text field. With sending an observation, the user can also add some extra material such as additional description, voice record, picture or location. Each observation refers to a Participant and enables change of time.

5.9 After-Action-Review (AAR)

Table 5.9: After Action Review

Item	Description
Short description	The After-Action Review (AAR) tool provides the possibility to collect data after a Trial has finished and analyse it. Its main purpose is to facilitate the evaluation of the trialled solutions, and to help the participants determine how well they functioned. It collects messages (exchanged during Trial), observation reports and takes screen-shots and these can then be deleted again (e.g. due to ethical considerations).
Who will use it	Facilitator
Main functions	The AAR tool will account for: - Storing relevant data - Reviewing a Trial completely or parts of it - Jump to specific point in time
Functions it does not do	The AAR tool will NOT account for: Changing the course of the Trial afterwards in any way
Links with other components	CIS, CSS, Scenario Manager, Observer Support Tool: The AAR tool uses the messages exchanged inside the CIS, CSS, Scenario Manager, and the observations to provide the review capability.
(Technical) conditions	The Security Service may have to grant the access to all secured topics to the AAR backend service to give the service the possibility to collect the data.
Reference to repository/details	github.com/DRIVER-EU/test-bed-admin [3]

5.10 Security Services

Table 5.10: Security Services

Item	Description
Short description	The Security Services provide access control enforcement on the DRIVER+ Test-bed, as well as support functions for identity and access management.
Who will use it	Directly: developers, Test-bed administrators. Indirectly: everyone using a solution connected to the Test-bed.
Main functions	The Security Services will account for: - Topic access policy enforcement; i.e. the Admin tool defines an access policy (set of access rules) per CIS topic, for the topics with confidentiality requirements, and delegates to the Security Services the enforcement of such policies in CIS. - SSL client certificate management, for managing (mostly issuing) SSL client certificates of CIS adapters for each tool connected to the Test-bed; these certificates are required for Test-bed level client authentication.
Functions it does not do	The Security Services will not account for: - The SSL authentication on CIS middleware (Kafka); this will rely on Kafka existing features, only the trusted CA certificate used in Kafka configuration comes from Security Services. (Not to be confused with the subsequent authorization phase that will be handled by Security Services, via Kafka extension.) - Protection, escrow or recovery of secret/private keys. Tools will have the possibility to have the Security Services generate key-pairs (with the certificates) for them, for Test-bed purposes only. However, the Security Services are not responsible for the protection, escrow or recovery of the generated private (or secret) keys in any way. If the certificate holder loses them, new ones – with a new certificate - will be generated. - Attack detection and/or mitigation.
Links with other components	- The Admin Tool consumes the Security Services for topic access policy configuration and enforcement in CIS. - The CIS trusts the Security Services CA for client authentication. - The CIS consumes the Security Services for topic access control.
(Technical) conditions	- Apache Kafka is needed. - Host OS should be Ubuntu 16.04 LTS 64-bit (or more recent), RAM >= 4GB, Filesystem ext4, disk space >= 40 GB, Gigabit LAN connection - Strongly recommended (but not mandatory): OS/applications should have access to a TPM preferably; else at least to high-quality entropy for cryptographic pseudo-random number generation.
Reference to repository / details	https://projectdriver.sharepoint.com/:p:/r/sites/DriverPlus/Documents%20partages/SP92%20-%20Testbed/WP923%20-%20Testbed%20infrastructure/Meetings/2018-02-12.14%20Vienna%20Development%20meeting%20February/Presentations/AccessControl-SecurityRoadmap-V3.pptx?d=w01f66b178f0143ad9eb3179fc3f8fc18&csf=1&e=VyUcKc [8] https://projectdriver.sharepoint.com/:w:/r/sites/DriverPlus/Documents%20partages/SP92%20-%20Testbed/WP923%20-%20Testbed%20infrastructure/Meetings/2018-02-12.14%20Vienna%20Development%20meeting%20February/DRIVER+%20F2F%20Minutes%20-%20Vienna%20-%202018-02-12.14.docx?d=w17d0df9098714babbeb746c0015a3a67&csf=1&e=UsdOiV [8]

5.11 Play service

Table 5.11: Play service

Item	Description
Short description	The Play service acts as a mini-scenario editor: it can either publish one message, or play a sequence of (timed) messages. Besides being useful for debugging, where a developer can replay a recorder scenario, it is also useful for testing solutions standalone, where the Play service can play a simple scenario. The recorded messages can be obtained using Landoop's Kafka Topics UI, which allows you to download all messages in a topic to a single JSON file with key-value messages.
Who will use it	Developers; Participants; Solution-owners/evaluators
Main functions	Publish a single recorded message, or a sequence of messages, to the Test-bed
Functions it does not do	Create messages from scrap, or record messages
Links with other components	It is connected to the Test-bed via an adapter, and it is related to the message injector
(Technical) conditions	Should run in a Docker environment, can get recorded messages from a mounted Docker volume
Reference to repository/details	github.com/DRIVER-EU/kafka-replay-service [9]

5.11.1 Extra notes

Related to this service is the open source csCOP (Common Operational Picture) tool, which can be used as a COP during a Trial, but which can also be used to test the messages published by simulators and solutions.

5.12 Message Injector

Table 5.12: Message Injector

Item	Description
Short description	The Message Injector is used for debugging purposes, similar to the developer application Postman, and it is used to inject messages manually into the Test-bed in order to generate a certain response.
Who will use it	Developers
Main functions	The main functionality is to inject (prepared or on the fly) messages into the Test-bed for testing purposes
Functions it does not do	
Links with other components	It will use one of the existing adapters, most likely, the Java one, to send its messages to the Test-bed
(Technical) conditions	Must run in a Docker environment

Item	Description
Reference to repository/details	

5.13 Data services

Table 5.13: Data services

Item	Description
Short description	The Data services are not a single service, but a group of non-essential, but very practical services to complement the Test-bed. Their main purpose is to share data to enrich the Trial, such as map data, height data, census layers, weather information, et cetera.
Who will use it	Trial staff
Main functions	Share map data, height data, vector data
Functions it does not do	Create or edit this data
Links with other components	Can be used by solutions as well as simulators as a data backend
(Technical) conditions	Must run in a Docker environment
Reference to repository/details	github.com/DRIVER-EU/test-bed-wms-service [10] github.com/DRIVER-EU/test-bed-mbtiles-service [11]

5.13.1 Extra notes

Related to these data services is the AVRO schemas repository, which contains all the schema's that have been used (so far).

5.14 Docker-composer

Table 5.14: Docker-composer

Item	Description
Short description	All of the Test-bed's core functionality run inside a Docker environment (*virtual machines*). The Docker-composer website allows you to select the Test-bed components you actually need, and it creates a dedicated Docker-compose file for you. This file can be easily run (`docker-compose up -d`), and the Test-bed is started using a single command, linking together services and data.
Who will use it	End users wishing to try out a solution, Developers and System Operators to setup the Test-bed
Main functions	Tie many different Docker images together
Functions it does not do	Create a Docker image for you
Links with other components	All Dockerized services, including the Dockerized solutions available in the Portfolio of Solutions

Item	Description
(Technical) conditions	Must run in a Docker environment
Reference to repository/details	driver-eu.github.io/docker-composer [12] github.com/DRIVER-EU/docker-composer [13]

6. Conclusions and way forward

The previous chapters described the Test-bed's environment, its high-level use-cases and its components and their reasons of existence. This chapter looks back at the previous chapters to draw some general conclusions about the Test-bed and look ahead on the development, implementation and functional review of the Test-bed. In the third and final section, links between WP923 (i.e. development and implementation of the Test-bed) and other SPs and WPs are described.

6.1 Overall conclusions on the Test-bed's requirements specifications

Chapter 4 described the high-level use-cases in which the components must function. The description of the environment the Test-bed is used in and descriptions of the components provide the specifications that apply as a "must" (i.e. a requirement) or as a "should" (i.e. not a hard requirement, but a wish that should be complied to as much as possible). These 'musts' and 'shoulds' are essential/important for a Test-bed to be useful and valuable in the Trials yet to come, being both the four Trials and Final Demo within the DRIVER+-project and future Trials outside this project.

This document does not describe how these requirements and wishes are to be fulfilled. Decisions on which technology is used to develop the components and why these off-the-shelf, innovative or newly created technologies are used, as well as choices made in the development of human-software interfaces are described in the deliverables D923.21, D923.22 and D923.23 Reference Implementation of the Test-bed (versions 1, 2 and 3).

Whether the specifications documented in this deliverable are fully valid, complete and/or feasible is yet to be shown. Conclusions about this can only be drawn when the components are developed and used in Trials: the proof is in the pudding. It is deliberately not foreseen that at this moment these specifications are fully correct and complete and actually will never fully be. The environments the Test-bed is to be used in are different per location and are dynamic over time in every potential location: technologies change, opinions of users differ and experiences grow. This is the main reason this requirements specifications document is a living document, allowing change when experiences in using the Test-bed are gathered. Furthermore, the development of the components is also staged, to offer several versions in time with a growing level of maturity and taking experiences gained during Trials in account (i.e. both on a quality level and on functionalities and interfaces available).

6.2 Envisioned development and implementation steps

The Test-bed will be developed in 3 phases:

Version 1 – to be used in Trials 1 and 2

This provides a first version of the CIS, CSS, logging within the CIS and CSS, adapters for solutions and simulators, Test-bed Admin tool, Observer Support Tool and the Play Service functioning as a first version of the Scenario Manager and Time Service (see also figure 4.2 with all components of the Test-bed). These components together form the functional core. The quality level of this version is that of a first prototype with a focus on functionality and stability needed to execute Trials 1 and 2. Potentially also a first version of the AAR component is developed. When specifically needed in the Trials, also a data-set of the fictive crisis environment(s) of Trial 1 or 2 is provided.

After use in Trials 1 and 2, this version is evaluated and the specifications will be updated regarding experiences gained in the use of the Test-bed. Updates on specifications and design choices will function as a base for developing version 2.

Version 2 – to be used in Trials 3 and 4 and Final Demo

This version comprises all components. The components should be reasonably stable with a quality level surpassing that of a first prototype. It should contain data-sets and basic scenarios that can be used for effectively connecting and testing (and assessing) solutions and simulators to the CIS and CSS and it should contain data-sets needed to execute Trials 3 and 4.

After use in Trials 3 and 4, this version is evaluated and the specifications will be updated regarding experiences gained in the use of the Test-bed. Updates on specifications and design choices will function as a base for developing version 3.

Version 3 – Final version

This version is the final version of all components. The quality level should surpass that of version 2. It should contain more data-sets and basic scenarios than available in version 2, to better support effectively connecting and testing (and assessing) solutions and simulators to the CIS and CSS and executing DRIVER+-external Trials.

This version comes with a final, online version of the specifications and design choices and documentation for implementation of the Test-bed outside DRIVER+. As these documents are available online via GitBook, people can still contribute to them even after the completion of the DRIVER+ project.

6.3 Dependencies of the Test-bed's development on other SPs and WPs

All Test-bed's components are developed in WP923. But this WP has links and even dependencies with other WPs and SPs. Especially the dependencies provide requirements for the development scheme and quality of the Test-bed.

WP922 – Evaluation of the Trial Guidance Methodology

The success of the Trials and thereby the Trial Guidance Methodology (i.e. how to prepare, execute and evaluate them) is under the influence of the successful use of the Test-beds components. For instance, the Observer Support Tool should make the observation easier and more detailed. So if these observations fail, the evaluation of a Trial is at risk. On a more operational level, the Observer Support Tool must provide the functionality to create observation templates complying to requirements for good observations set by the Trial Guidance Methodology.

T924.1 – Training Module

The Training Module is primarily aimed at people who want to do a Trial. This Training Module will contain a section explaining the Test-bed and what it can offer (i.e. of which components it consists of and for what these components are intended to be used). Although there is no hard dependency between WP923 and T924.1, the Training Module will highly benefit from images or even videos showing interfaces of some components with understandable, applicable (demo-) Trial content.

SP93 – Development of Portfolio of Solutions and Guidance Tool

Descriptions of the Test-bed's components should also be taken up in the Portfolio of Solutions. This provides an accessible entrance to what the Test-bed comprises of, also for people outside DRIVER+.

The Guidance Tool should contain forms to enter scenario information and describe injects in these scenarios. The template of injects should be aligned with the definition of an inject in the Scenario Manager and the CSS.

SP94 and T924.2 – support to and execution of Trials and Final Demo

The successful execution of Trials of course is highly dependent on timely availability of the components needed in each Trial. The quality of these components needs to be fit-for-use. Fit-for-use means the quality of a component should not jeopardize the success of the Trial (e.g. should not cause a crash of the Test-bed or a Solution), but it does not mean that in a Trial every component must have the quality level of a final version.

For an effective (technical) support to Trials in T924.2, it is essential that the Test-bed can be easily (partly) implemented at a Trial Owner's location, but also at Solution providers for linking up their Solution to the Test-bed (if needed and possible). The Docker environment supports this easy installation, but this dependency also requires that each component is well documented and potential bugs encountered can be quickly pinpointed and resolved.

SP94 also deals with the evaluation of the TGM, the Test-bed and the supportive tools GT, TAP and PoS. Further development of the Test-bed is thus partly dependent on the outcome of the Test-bed's evaluation after each Trial.

WP954 – Test-bed's sustainability

As already described, the Test-bed must be developed such that it can also be used outside/after DRIVER+. The development and implementation of the Test-bed has a strong link with the sustainability tasks in WP954. It can even be stated that the sustainability of the Test-bed and thereby results of the DRIVER+ project (highly) depends on development choices made in the development of the components.

References

1. Online living version of this document D923.11 on GitBook:
   driver-eu.gitbooks.io/Test-bed-specification/content/

2. Agile software development Wikipedia definition:
   en.wikipedia.org/wiki/Agile_software_development

3. Test-bed manager (Admin tool) software code repository:
   github.com/DRIVER-EU/test-bed-admin

4. Scenario Manager software code repository:
   github.com/DRIVER-EU/scenario-editor

5. Time Service software code repository:
   github.com/DRIVER-EU/test-bed-time-service

6. Observer Support Tool software code repository:
   github.com/DRIVER-EU/ost

7. Observer Support Tool detailed specifications and design:
   driver-eu.gitbooks.io/specification-of-the-online-observer-support-tool

8. Security services design decisions:
   https://projectdriver.sharepoint.com/:p:/r/sites/DriverPlus/Documents%20partages/SP92%20-%20Testbed/WP923%20-%20Testbed%20infrastructure/Meetings/2018-02-12.14%20Vienna%20Development%20meeting%20February/Presentations/AccessControl-SecurityRoadmap-V3.pptx?d=w01f66b178f0143ad9eb3179fc3f8fc18&csf=1&e=VyUcKc
   https://projectdriver.sharepoint.com/:w:/r/sites/DriverPlus/Documents%20partages/SP92%20-%20Testbed/WP923%20-%20Testbed%20infrastructure/Meetings/2018-02-12.14%20Vienna%20Development%20meeting%20February/DRIVER+%20F2F%20Minutes%20-%20Vienna%20-%202018-02-12.14.docx?d=w17d0df9098714babbeb746c0015a3a67&csf=1&e=UsdOiV

9. Play service software code repository:
   github.com/DRIVER-EU/kafka-replay-service

10. WMS service software code repository:
    github.com/DRIVER-EU/test-bed-wms-service

11. MBtiles service software code repository:
    github.com/DRIVER-EU/test-bed-mbtiles-service

12. DRIVER+ Test-bed Docker composer website:
    driver-eu.github.io/docker-composer

13. DRIVER+ Test-bed Docker environment software code repository:
    github.com/DRIVER-EU/docker-composer

14. *DRIVER+ (2018). D922.21 - Trial guidance methodology and guidance tool specifications (version 1)*

15. *DRIVER+ (2018). D923.21 - First release of the Test-bed reference implementation*

16. *CRISIS-project (2010 – 2013)* coordinated by Middlesex University, which received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° FP7-242474

Annexes

Annex 1 – DRIVER+ Terminology

In order to have a common understanding within the DRIVER+ project and beyond and to ensure the use of a common language in all project deliverables and communications, a terminology is developed by making reference to main sources, such as ISO standards and UNISDR. This terminology is presented online as part of the Portfolio of Solutions and it will be continuously reviewed and updated[5]. The terminology is applied throughout the documents produced by DRIVER+. Each deliverable includes an annex as provided hereunder, which holds an extract from the comprehensive terminology containing the relevant DRIVER+ terms for this respective document.

Terminology	Definition	Comment
Crisis Management	Holistic management process that identifies potential impacts that threaten an organisation and provides a framework for building resilience, with the capability for an effective response that safeguards the interests of the organisation's key interested parties, reputation, brand and value-creating activities, as well as effectively restoring operational capabilities.	Note to entry: Crisis management also involves the management of preparedness, mitigation, response, and continuity or recovery in the event of an incident, as well as management of the overall programme through training, rehearsals and reviews to ensure the preparedness, response and continuity plans stay current and up-to-date.
Evaluation	Process of estimating the effectiveness, efficiency, utility and relevance of a service or facility	
Observation	Method of data collection in which the situation of interest is watched and the relevant facts, actions and behaviours are recorded [observer participant who witnesses the exercise while remaining separate from exercise activities]	
Portfolio of Solutions (PoS)	A database driven website that documents the available Crisis Management solutions. The PoS includes information on the experiences with a solution (i.e. results and outcomes of Trials), the needs it addresses, the type of practitioner organisations that have used it, the regulatory conditions that apply, societal impact consideration, a glossary, and the design of the Trials.	
Scenario	Pre-planned storyline that drives an exercise; the stimuli used to achieve exercise objectives [pre-planned storyline that drives an exercise, as well as the stimuli used to achieve exercise project performance objectives]	

[5] Until the Portfolio of Solutions is operational, the terminology is presented in the DRIVER+ Project Handbook and access can be requested by third parties by contacting coordination@projectdriver.eu.

Terminology	Definition	Comment
Test-bed	The software tools, middleware and methodology to systematically conduct Trials and evaluate solutions within an appropriate environment. An "appropriate environment" is a testing environment (life and/or virtual) where the trialling of solutions is carried out using a structured, all-encompassing and mutual learning approach. The Test-bed can enable existing facilities to connect and exchange data, providing a pan-European arena of virtually connected facilities and crisis labs where users, providers, researchers, policy makers and citizens jointly and iteratively can progress on new approaches or solutions to emerging needs.	
Training	Activities designed to facilitate the learning and development of knowledge, skills, and abilities, and to improve the performance of specific tasks or roles.	
Trial	An activity for systematically finding and testing valuable solutions for current and emerging needs in such a way that practitioners can do this in a pragmatic yet systematic way.	
Trial Action Plan (TAP)	The main Trial planning document, facilitating collaborative planning and supporting combined execution. It covers all areas related to the Trial organization and will be used to record efforts, circulate decisions and assess progress.	
Trial Guidance Methodology	A structured approach from designing a Trial to evaluating the outcomes and identifying lessons learned.	