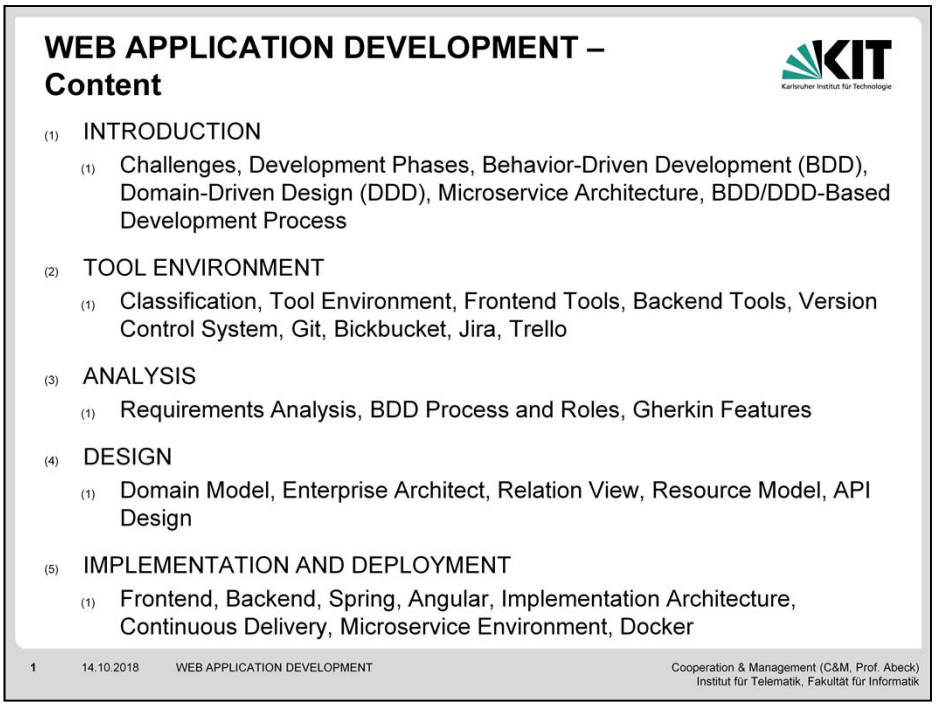


In this course unit [CM-W-WEB] a structured software development process of web applications is illustrated. The application development makes use of current software development concepts, such as Behavior-Driven Development (BDD), Domain-Driven Design (DDD), microservice architectures including a systematic design of the web Application Programming Interfaces (API) of the microservices.

(1) An overview of the relevant development concepts is given and the application development process using these concepts is introduced. The core concepts and technologies are illustrated with the example of a web application, by name TodoListManagement (TLM), is introduced.

(2) The development of advanced web application requires a complex tool environment. The organization of software development projects is supported by a specific set of project management and version control tools which are described in more detail in this chapter.

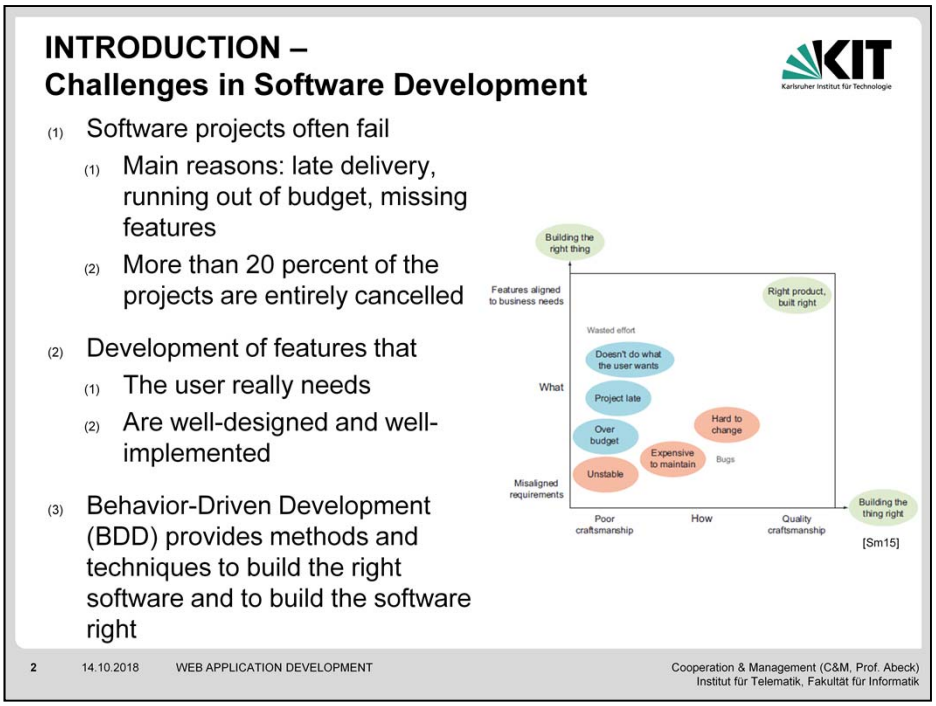
(3) This chapter deals with the analysis, the first phase of the development process, in which the requirements are specified. In the approach described in this course unit so-called Gherkin features are used. Those features are the central artifact in the case of Behavior-Driven Development (BDD).

(4) The domain model and the APIs are central artifacts of the subsequent design phase. In the domain model established according to the principles of Domain-Driven Design (DDD), the (structural and functional) knowledge needed to implement the software system is formally captured. A method by which the APIs are systematically derived from the domain model is introduced.

(5) In the implementation phase the functionality as is was specified in the analysis and design phase is coded based on a mircroservice architecture and frameworks supporting the frontend and backend implementation. In the deployment phase the tested application is deployed in a Docker-based execution environment.

API	Application Programming Interface
BDD	Behavior-Driven Development
C&M	Cooperation & Management
DDD	Domain-Driven Design
KIT	Karlsruhe Institute of Technology
TLM	TodoListManagement

[CM-W-WEB] Cooperation & Management: WEB APPLICATION DEVELOPMENT, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*



The motivation behind Behavior-Driven Design (BDD) is to build and deliver better software [Sm15:3].

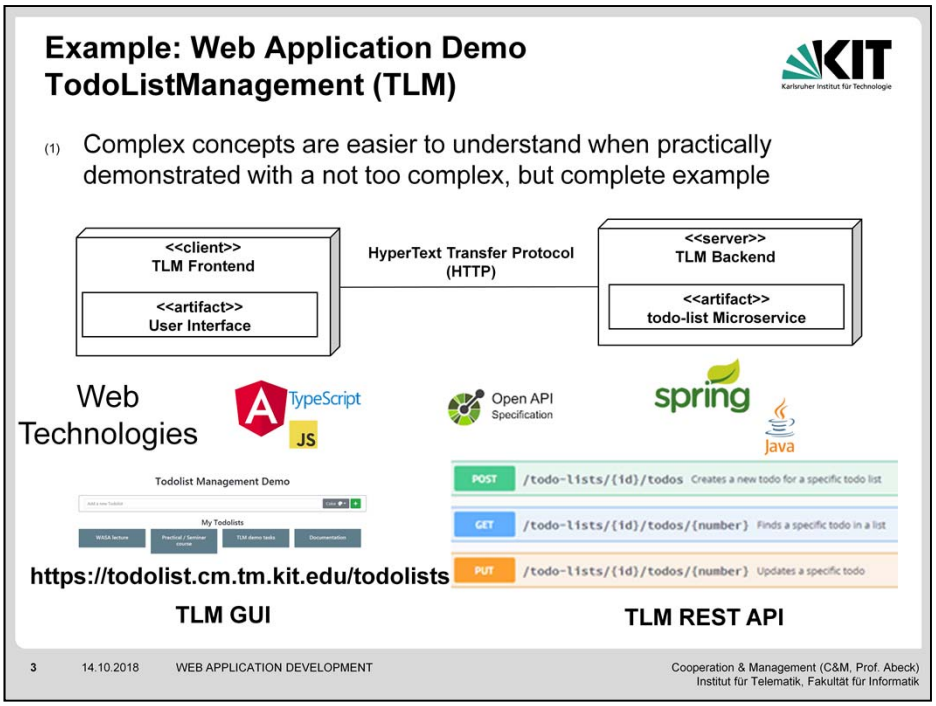
(1) According to a number of studies, nearly half of all software projects fail to deliver in some significant way.
(1.1) In a CHAOS Report 2011 published by the Standish Group in 42 % of the software projects one or more of these problems occurred.
(1.2) This results in billions of dollars in wasted effort.

(2) The two goals center around the questions of WHAT to develop and HOW to develop.
(2.1) This goal describing the WHAT is shown by the vertical (y) axis in the figure.
(2.2) This goal describing the HOW is shown by the horizontal (x) axis in the figure.

(3) BDD encourages business analysts, software developers, and testers to collaborate more closely by enabling them to express requirements in a more testable way, in a form that both the development team and business stakeholders can easily understand.

BDD Behavior-Driven Development

[Sm15] John Ferguson Smart: BDD in Action – Behavior-Driven Development for the whole software lifecycle. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD*



Throughout this course unit, the TodoListManagement (TLM) web application by which a user can manage his/her todos in lists, is used to demonstrate the technologies and methods of web application development. This application is also in the focus of the initial phase of the practical and seminar course offered in parallel to the lecture. A detailed technical description can be found in [CM-W-TLM].

(1) To be able to develop complex web applications it is not only necessary to (theoretically) understand the quite complex concepts, but also to (practically) apply them. With the TLM example the practical use of the concepts is demonstrated.

(<<client>> TLM Frontend, HTTP, <<server>> TLM Backend) Since TLM is a web application it consists of a web client and a web server which communicate via the Internet application protocol HyperText Transfer Protocol (HTTP).
The diagram describes the physical architecture of the TLM application and uses modeling elements specified by the Unified Modeling Language (UML). It is a so-called deployment diagram by which the physical systems (in this case, TLM Frontend and TLM Backend) are modeled as specific UML symbol called nodes.

(Web Technologies) Technologies play an important role in web application development. According to the physical architecture of a web application, frontend and backend technologies as well as technologies concerning the interface (Application Programming Interface, API) can be distinguished)

(A, TypeScript, JS) The "A" logo stands for Angular which is a leading frontend technology. Angular uses the script language TypeScript which is based on Java Script (see JS logo).

(TLM GUI) This is a cutout of the TLM Graphical User Interface (GUI) which is implemented with Angular.

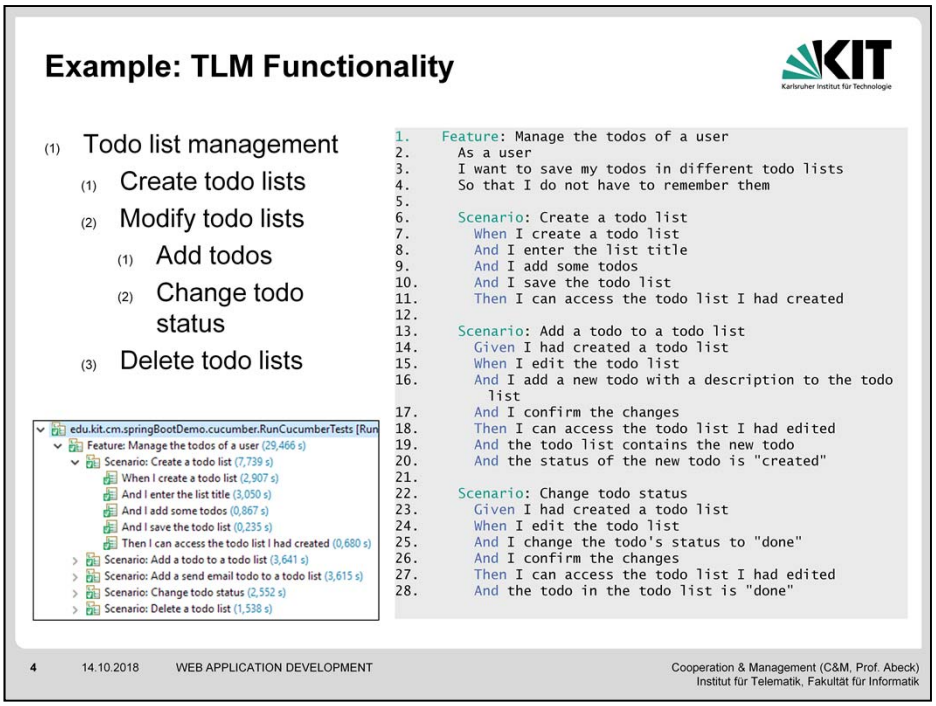
(Spring, Java) Spring is one of the leading Java-based backend technologies.

(OpenAPI Specification) The OpenAPI Specification is a manufacturer-independent defacto standard for the definition of REST APIs.

(REST API) (TLM REST API) REST stands for Representational State Transfer which is the most widely used concept for the specification of web APIs.
(POST /todo-lists/{id}/todos ..., GET /todo-lists/..., PUT ...) This is an excerpt of the REST API which is provided by the todo-list microservice. Each REST operation is a specific HTTP method, such as GET or POST. For example, the REST operation "POST /todo-lists/{id}/todos" has a parameter which describes the todo which is added to the todo list identified by "id".

API	Appliocation Programming Interface
HTTP	HyperText Transfer Protocol
JS	JavaScript
REST	Representational State Transfer
TLM	TodoListManagement
UML	Unified Modeling Language

[CM-W-TLM] Cooperation & Management: TLM PRACTICAL COURSE AND EXERCISES, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-2.Leitfäden*



The TodoListManagement (TLM) web application serves as an example to demonstrate the concepts of web application development introduced in this course. One of these concepts is the Behavior-Driven Development (BDD) by which the requirements of a software system are specified using features described in the language Gherkin.

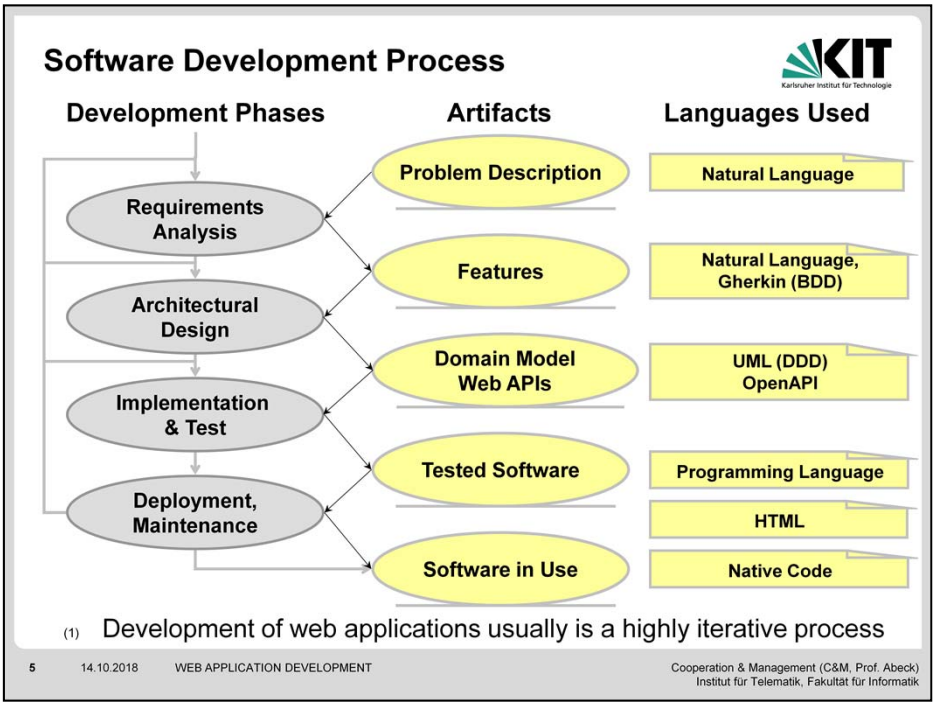
(1) The sample project is a simple todo administration software.
(1.1) It should be possible to create a new todo list and add it to the existing lists of todos. Lists can have a title and an unlimited number of todos.
(1.2) Existing lists can be edited.
(1.2.1) The title can be changed or a new todo can be added to the list.
(1.2.2) It is also possible to mark a todo as done.
(1.3) Another requirement is to delete existing lists.

(Feature: Manage the todos of user) The requirements on the software system are presented as features which are written in a language called Gherkin. Most of the feature description is written in a natural language. In order to allow interpretation with the computer, Gherkin prescribes some keywords (in the example: Feature, Background, Scenario, Given When, Then And) which must be considered. Gherkin is used by the Cucumber framework [Cuc-Ger].

(edu.kit.cm.springBootDemo.cucumber.RunCucumberTests) In all later phases of the software development process the definition can be used to test the software. In this way, it is possible to test whether a software meets the requirements and, in case of changes to the code, to check whether all requirements are still being met. The tests are started as Junit tests and can be integrated into all existing tools [Jun-Ju4].

BDD	Behavior-Driven Development
TLM	TodoListManagement

[Cuc-Ger] Cucumber: Gherkin. *https://github.com/cucumber/cucumber/wiki/Gherkin*
[Jun-Ju4] Junit: Junit 4. *https://junit.org/junit4/*



The figure introduces the different phases that have to be carried out to develop software systems, esp. web applications. Besides the programming language, natural language and several description languages are needed to develop a web application in a sound and systematic way.

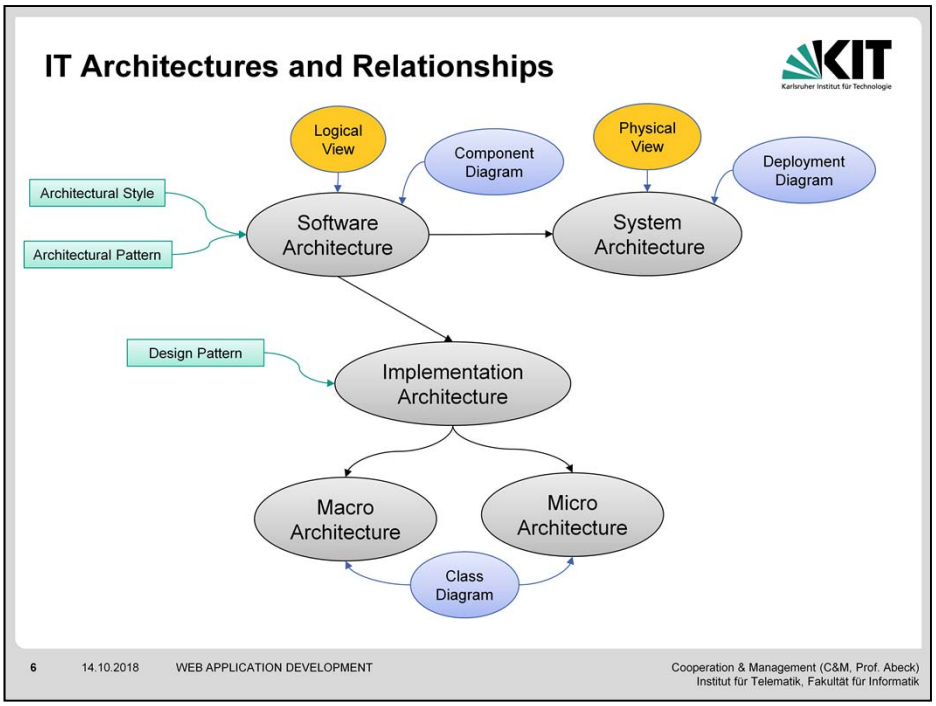
(1) Iterative in this context means that the phases are not completely passed in a sequential order but jumps to a the preceding phase are the usual case.

(Feature, Domain Model) Features are the main artifacts from Behavior-Driven Development whereas the domain model is the main artifact introduced by the Domain-Driven Design (DDD).

(Unified Modeling Language, eXtensible Markup Language, Hypertext Markup Language) are widely used description languages in the development process of web applications. Gherkin is a language which is used in the BDD approach.

(OpenAPI) This is a standardized language by which an API (Application Programming Interface) of a microservice based on REST (REpresentational State Transfer) can be specified.

API	Application Programming Interface
BDD	Behavior-Driven Development
DDD	Domain-Driven Design
REST	REpresentational State Transfer
UML	Unified Modeling Language
XML	eXtensible Markup Language



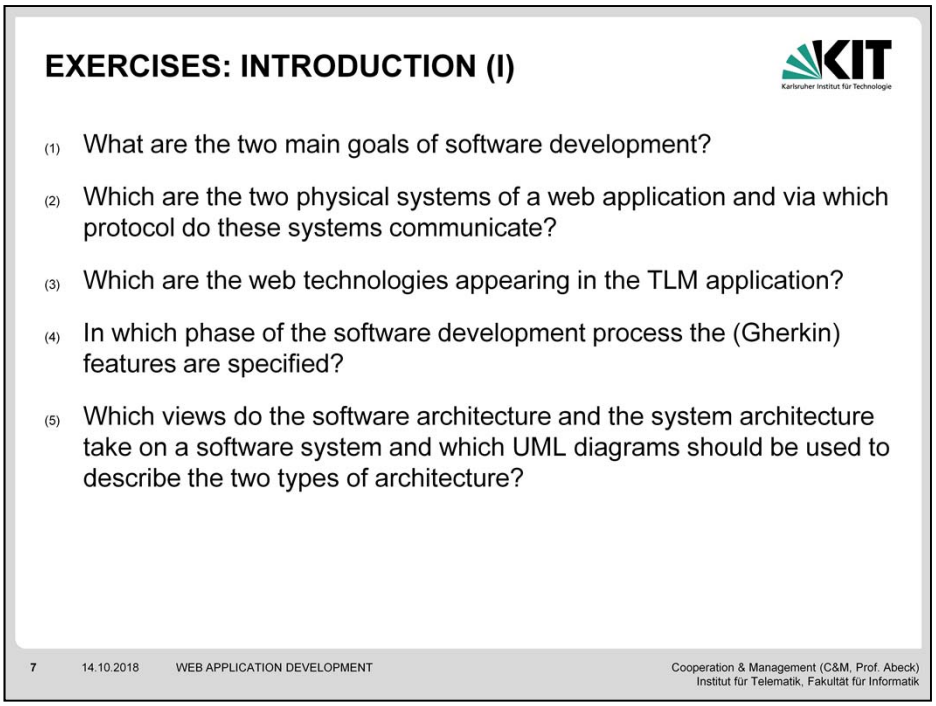
This figure illustrates the various architectural concepts and their relationships. It should be noted that no standardized definitions exist. For example, [CMU-SA] lists more than 30 definitions for software architecture.

(Software Architecture) The software architecture takes a logical view on the system by dividing it into logical components.
(System Architecture) The system architecture takes a physical view on a system and describes its structure which consists of network and hardware components [VA+08]. Furthermore, their properties and relationships to each other as well as to their environment and other systems are shown. A possible modeling type is the deployment diagram.
(Implementation Architecture) The implementation architecture describes the structure of the system from a technical point of view [RM+06]. This includes packages, libraries and frameworks.
(Macro Architecture, Miocro Architecture) The macro architecture is used to describe the principles of the subsystems, e. g. micoservices or modules, on a black box level. The micro architecture deals with the internal structure of a single subsystem [Ot13].

(Architectural Style) Architectural styles [RH08] are application-independent solution principles that are used throughout the project. These can be assigned to the following categories: communication (e.g. message bus oriented), deployment (e.g. N-tier architecture) and structure (e.g. layer architecture).
(Design Pattern) Design patterns [RH08] provide program code-specific patterns (e.g. visitor pattern) for recurring problems.
(Architectural Pattern) Architectural patterns [RH08] are solutions to recurring problems which, unlike design patterns, affect several architectural elements.

(Component / Deployment / Class Diagram) Deployment View) The Unified Modeling Language (UML) provides different types of diagrams to specify the different types of architecture as models.

[CMU-SA] CMU: Software Architecture Getting Started Glossary Modern Software Architecture Definitions. *https://www.sei.cmu.edu/architecture/start/glossary/moderndefs.cfm*
[Ot13] Otto: Architekturprinzipien – dev.otto. *https://dev.otto.de/2013/04/14/architekturprinzipien-2/*
[RM+06] John Reekie, R. J. Mcadam, Rohan McAdam: A Software Architecture Primer, Angophora Press, 2006.
[RH08] Ralf Reussner, Wilhelm Hasselbring: Handbuch der Software-Architektur, dpunkt Verlag, 2008.
[VA+08] Oliver Vogel, Ingo Arnold, Arif Chughtai, Edmung Ihler, Timo Kehrer, Uwe Helog, Uwe Zdun: Software-Architektur: Grundlagen – Konzepte –Praxis, page 48ff. Springer Verlag, 2008.



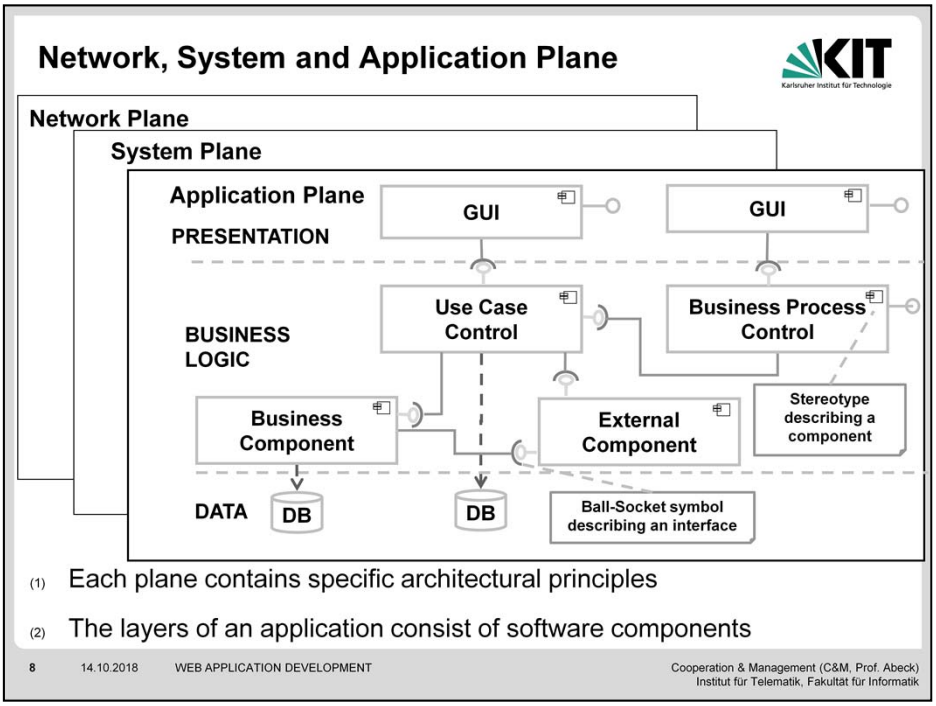
(1) ++Challenges in Software Development++
- Goal 1: To develop the right software (WHAT should be developed?)
- Goal 2: To develop the software right (HOW should the software be developed?)

(2) ++Example: Web Application Demo TodoListManagement (TLM)++
- Physical systems: Web client (frontend), web server (backend)
- HTTP (HyperText Transfer Protocol) which is an Internet application protocol.

(3) ++Example: Web Application Demo TodoListManagement (TLM)++
- Angular: A leading frontend technology
- TypeScript: Used by Angular and based on Java Script
- Spring, Java: Spring is one of the leading Java-based backend technologies.
- OpenAPI Specification: A manufacturer-independent defacto standard for the definition of REST APIs.

(4) ++Software Development Process++
- In the analysis phase
- Remark: Features are described by using the language Gherkin. They are the central artifact of the Behavior-Driven Development (BDD).

(5) ++Types of Architectures and Relationships++
- Software architecture: logical view, UML component diagram
- System architecture: physical view, UML deployment diagram



(1) The fundamental principles of each plane (e.g. layering) are described in the following.

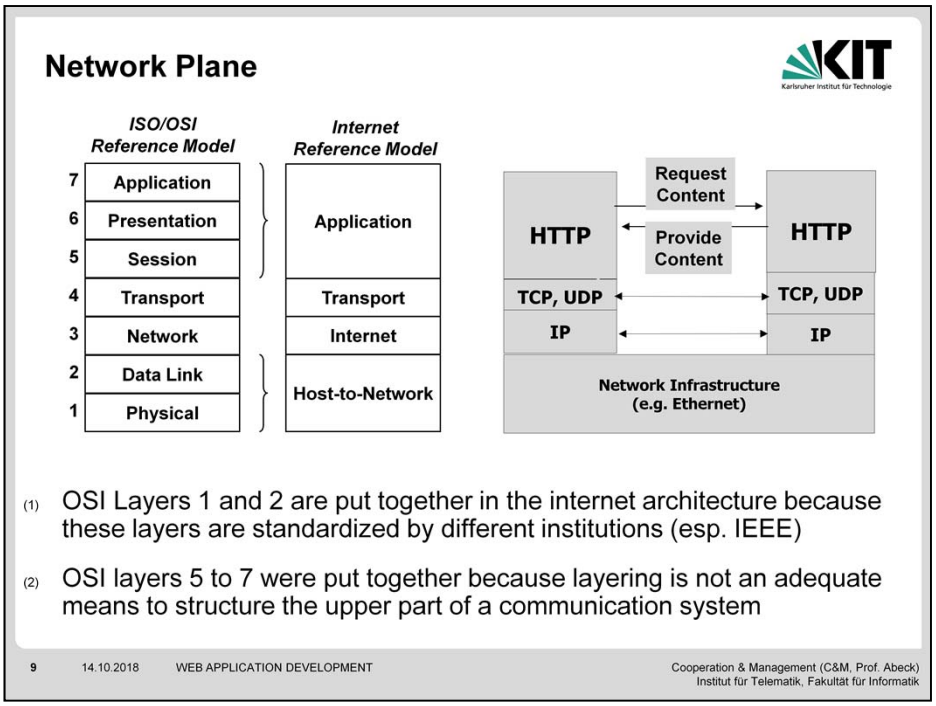
(Network Plane) Web applications run on several independent systems which heavily use Internet technologies for communication purposes. The network plane consists of communication layers. In each such layer the communication is organized by standardized protocols (e.g. IP, HTTP).

(System Plane) Internet protocols (such as the web protocol HTTP) are based on a client-server principle. The system plane is characterized by different operating systems leading to heterogeneous IT infrastructures.

(Application Plane) Web-based applications are located on the application plane. The term "web-based" especially means that the presentation layer is implemented by a standard web browser.
(DATA, BUSINESS LOGIC, PRESENTATION) The logical separation concerns the different (logical, conceptual, architectural) aspects a distributed application has to cover. There are three aspects – data, function (or business) and presentation – which can be found in every application.

(2) The Unified Modeling Language (UML) is an adequate language to graphically describe the architecture of a software application. Modeling is necessary in order to develop the application in a structured and systematic way.
(Stereotype describing a component) A component in UML is modeled as a rectangle with a specific icon in the right upper corner. This icon is a graphical representation of a stereotype which is further defined in the UML meta model. An equivalent textual representation of this stereotype defining a component is <<component>>.
(Ball-Socket symbol describing an interface) The ball represents the provided part and the socket represents the required part of the component interface.

DB	Data Base
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
UML	Unified Modeling Language



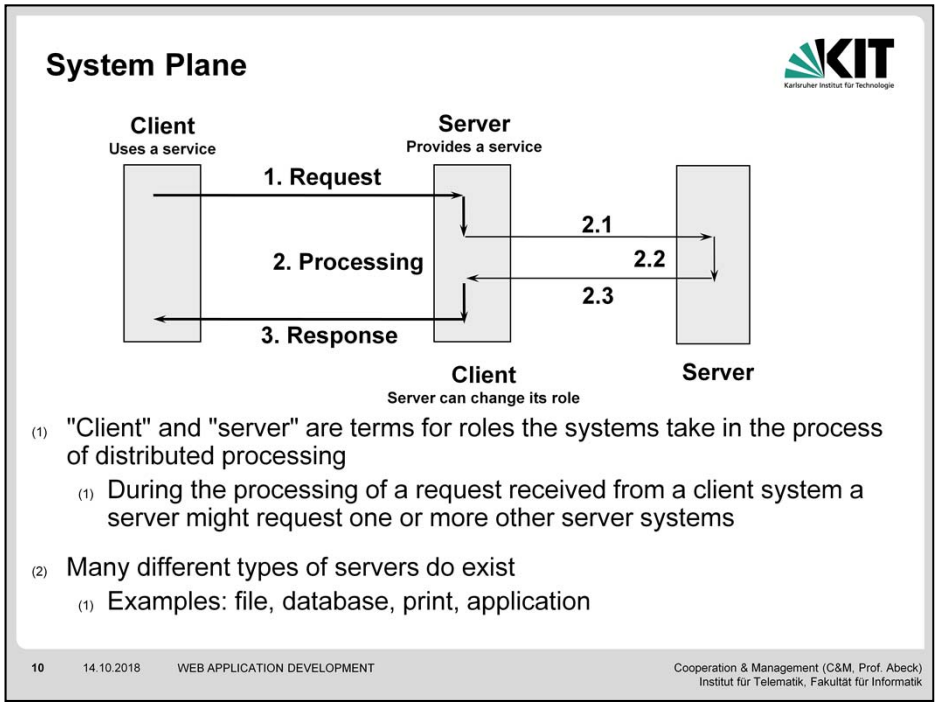
There are so-called communication reference models that define the functionality of the layers that build the network plane. Two important models are shown on this page [KR05, Ta06].

(HTTP, Request Content, Provide Content) The HyperText Transfer Protocol (HTTP) is an Internet application protocol which allows a HTTP client to request and also change content residing at an HTTP server by offering different HTTP operation, such as GET or POST.

(1) (2) The merging of the two lower layers in the Internet architecture has "organizational" reasons whereas the missing layering in the application system is conceptually motivated.

HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Standards Organization
OSI	Open Systems Interconnection
TCP	Transmission Control Protocol

[KR05] James F. Kurose, Keith W. Ross: Computer Networking – A Top-Down Approach Featuring the Internet, Pearson Education, 2005.
[Ta06] Andrew .S. Tanenbaum: Computer Networks, Prentice Hall, 2006.

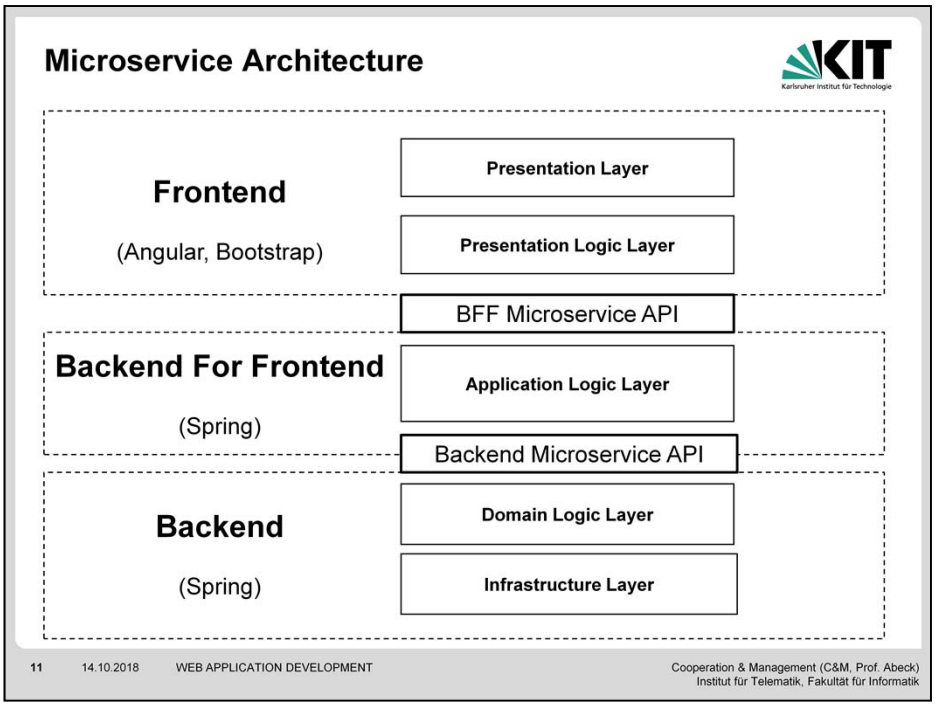


On the system plane of a web application the involved systems take the roles of clients and servers.

(1) Processing is usually organized according to a client-server principle: The client asks the server to do some processing.

(2) Central tasks the server systems listed in the slide have to provide, are:
- File server: Upload and download of files
- Database server: SQL-based search queries
- Print server: offers functionality specific to the print service, such as queuing or prioritizing of print orders
- Application server: contains executable code specific to one or more applications

SQL Standard Query Language



A microservice architecture is located on the application plane as it is introduced in ++Planes and Three-Layer Application Architecture++. It consists of three parts (backend, backend-for-frontend BFF, frontend) and two types of application programming interfaces (BFF) via which these parts are interacting (backend microservices API, BFF microservices API).

(Presentation Layer) This layer renders the UI elements in the browser. Technologies that support the implementation are Angular and Bootstrap.

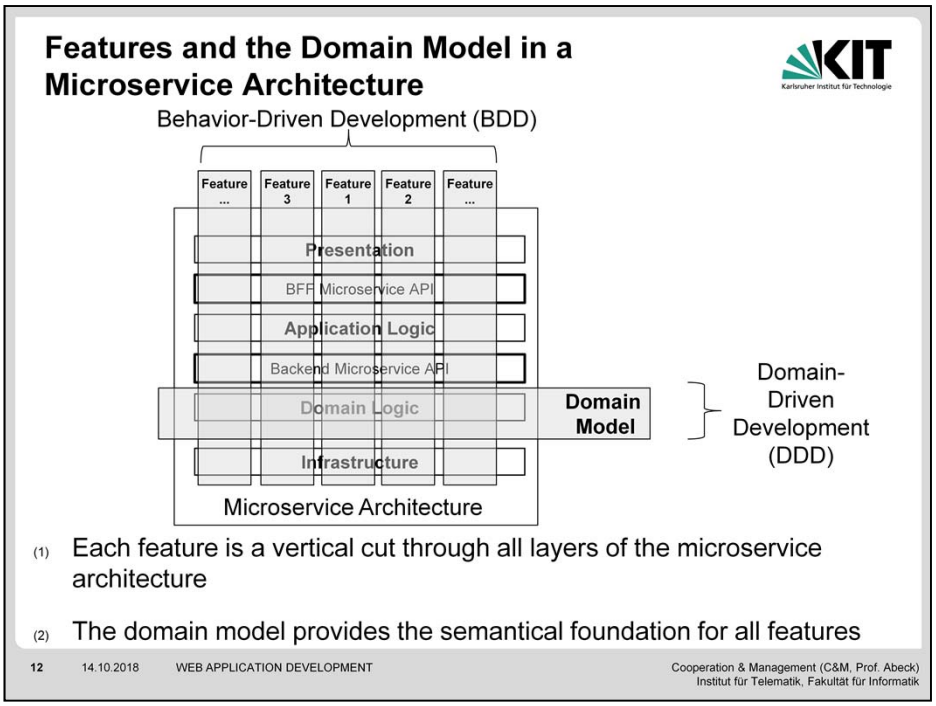
(Presentation Logic Layer) The presentation includes a logic which controls the interaction with the BFF microservice API.

(Application Logic Layer) This layer realizes the orchestration of the backend microservices in order to provide the BFF microservices required by the frontend. A technology that supports the implementation of this functionality is Spring.

(Domain Logic Layer, Infrastructure Layer) These layers implement the backend services which are CRUD operations on the domain objects.

In contrast to a traditional three-layer application architecture the business logic layer in a microservice architecture is split into two layers, the domain logic layer and the application logic layer. The reason for that is to promote the reuse of business logic functionality by distinguishing between application-agnostic (= domain logic) and application-specific (= application logic) functionality.

API	Application Programming Interface
BFF	Backend For Frontend
CRUD	Create, Read, Update, Delete

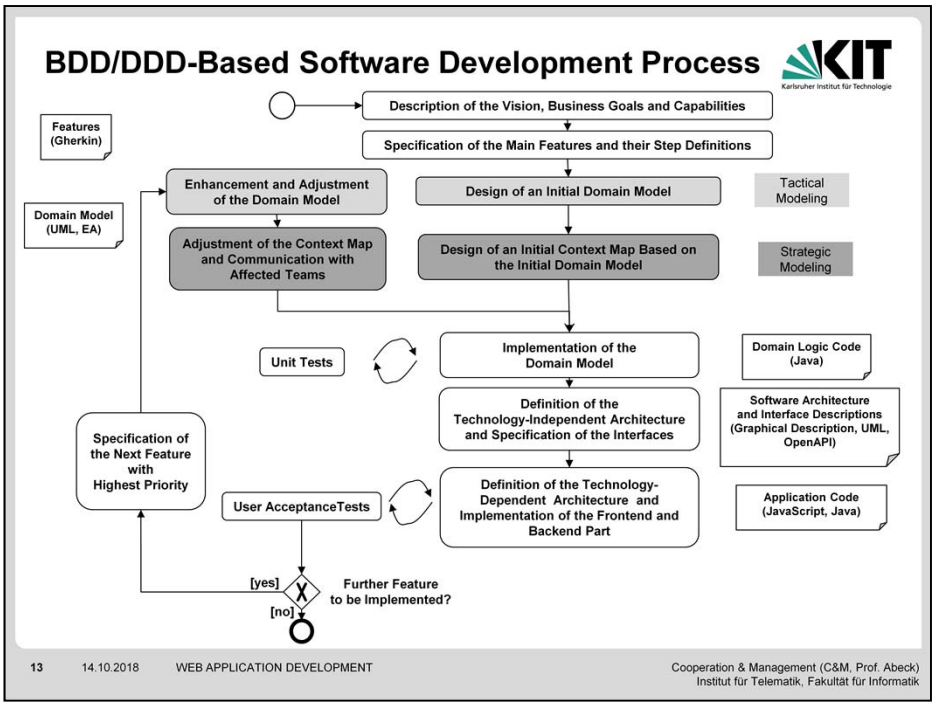


The software development process applied by C&M combines the concepts of Behavior-Driven Development (BDD) an Domain-Driven Design (DDD). Both concepts provide complementary contributions to the layered microservice architecture as the figure illustrates.

(1) An implemented feature can be seen as a deployable increment of the software system.
(Feature1, Feature 2, ...) The ordering of the features in the figure implies that the first feature should cover the core functionality of the software system.

(2) The domain model makes sure that the static and dynamic domain knowledge is consistently used by each feature. This ensures that the features build a consistent whole although each feature is developed and deployed independently from other features.

BDD	Behavior-Driven Development
DDD	Domain-Driven Design



The BDD/DDD-based software development process illustrated on this page is an agile approach in which the software system is built feature by feature. As the activity diagram shows the feature-driven cycle starts with the specification of the next feature with the highest priority and ends up with the test of the implemented feature.

(Description of the Vision, Business Goals and Capabilities) Before developing an application the vision and goals that should be achieved should be clear. Business goals describe the benefits that the business gets from of the system. The capabilities describe the functionality that is needed to achieve the business goals in a non-technical way. Features are derived from the business capabilities.

(Specification of the Main Features and their Step Definitions) The requirements elicitation is conducted by applying the BDD concept. It is recommended to reflect about the domain of the software before the first feature is specified. If a domain model for this domain already exists it necessarily should be taken into account to make sure that the domain logic is consistently used by the software system to be developed.

(Design of an Initial Domain Model) (Designing an Initial Context Map Based on the Initial Domain Model) Feature specification is an analysis activity and domain model specification is a design activity. The goal of this combined approach is to make clear which domain knowledge is included in each specified feature. After the initial domain model is created, the first version of the context map is derived.

(Implementation of the Step Definitions and the Domain Model) In this step the domain logic layer is implemented according to the specification of the domain model.

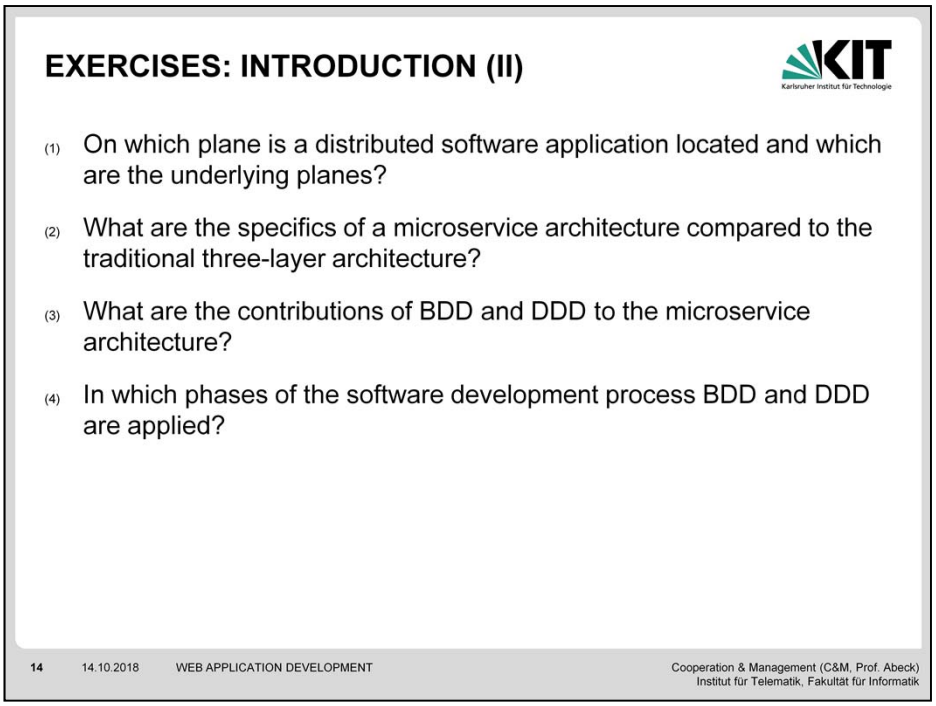
(Definition of the Technology-Independent / Dependent Architecture) The technology-independent description is based on the specification of (partly web) APIs whereas in the technology-dependent description the used frontend and backend frameworks appear.

(Unit Tests, Test of the Feature) Testing both on the technical unit level and the user-oriented feature level is an important characteristic of the development process.

(Specification of the Next Feature with Highest Priority) If there are further features available, the features with the highest priority are specified and implemented.

(Adjustment of the Domain Model) A new feature may lead to new insight. This insight can lead to changes in the domain model. In some cases it is necessary to change the relation between bounded contexts. This leads to changes that are considered in the next step.

(Adjustment of the Context Map) After each cycle, the context map may need some adjustments. In early stages, there may be more adjustments than in a later stage. Depending on the strategy used, affected teams need to communicate the necessary changes. The context map shows the teams that are involved.

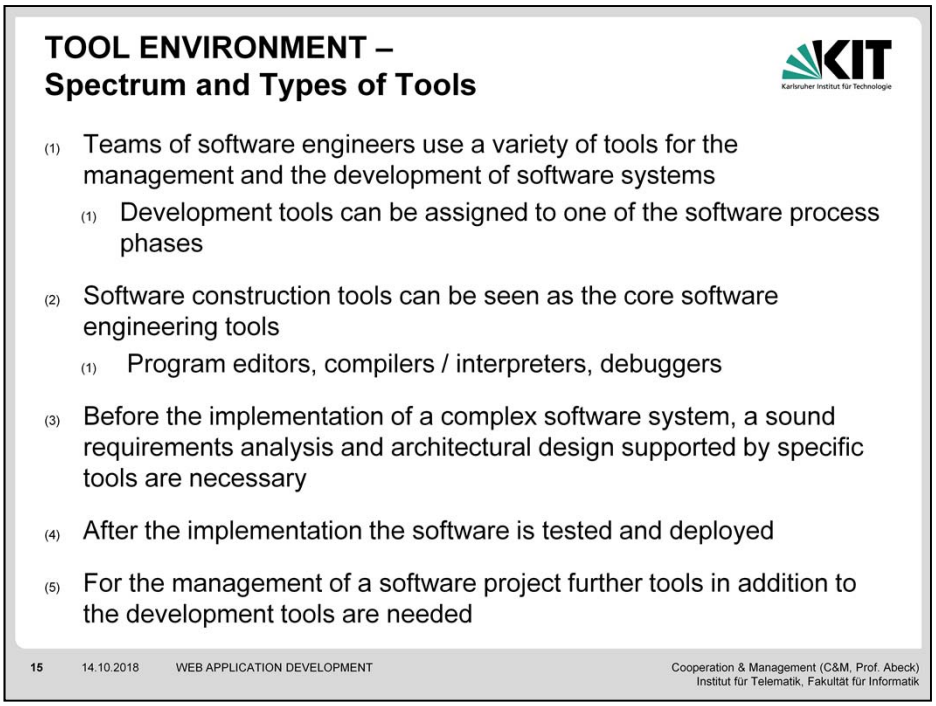


(1) ++Planes and Three-Layer Architecture++
- Application plane
- System plane, network plane

(2) ++Planes and Three-Layer Architecture++, ++Microservice Architecture++
- Separation of the business logic layer (middle layer) into two layers, the application layer and the domain layer
- Explicit introduction of two APIs, the Backend Microservice API and the Backend for Frontend Microservice API

(3) ++Features and the Domain Model in a Microservice Architecture++
- BDD: Each feature is a vertical cut through all layers of the microservice architecture
- DDD: The domain model is implemented in the domain logic layer and provides the semantical foundation for all features

(4) ++BDD/DDD-Based Software Development++
- BDD: Influences the analysis by providing the specification based on features. In addition, a feature defines the software increment which is implemented in an agile way.
- DDD: The domain model is the central design artifact of the development process.



(1) On the one hand, the tools support the software engineers and make software development easier and more efficient. On the other hand, the broad spectrum of different tools makes software development more complex since the correct use of the tools is a time-consuming challenge for each developer.
(1.1) The phases are: analysis, design, implementation, testing, deployment. Management tools cannot be assigned to one specific phase are which are used by the teams for communication purposes and for the organization of the software project.

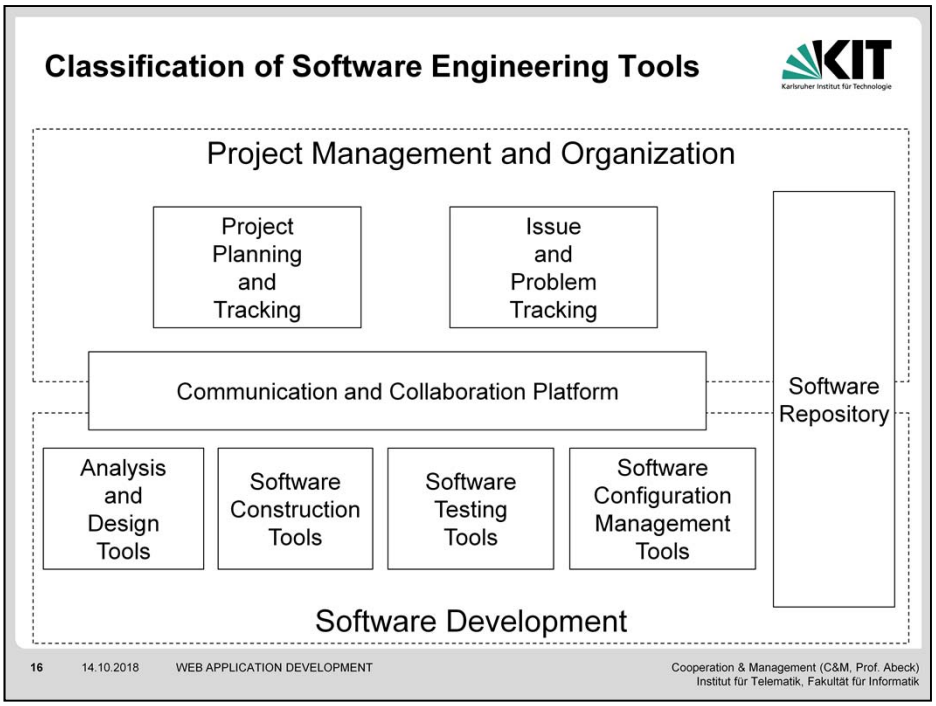
(2) This type of development tools is indispensable for the development of software.
(2.1) Construction tools are used in the implementation phase.

(3) An example of a software requirements tool is Cucumber which supports the approach of behavior-driven development (BDD). Examples of software design tools are UML modeling tools or tools to specify the APIs (application programming interface) of a software architecture.

(4) Software testing tools include test generators, test execution frameworks, test evaluation, test management and performance analysis. Tools which support the deployment are called software configuration management tools [Ca01].

(5) The main support provided by this type of development tools is project planning and tracking of tasks.

[Ca01] David Carrington: Software Engineering Tools and Methods, IEEE – Trial Version 1.00, 2001. *https://pdfs.semanticscholar.org/100c/1b90a8870dd256ceb98c6a831bba3dc9cf92.pdf, https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*



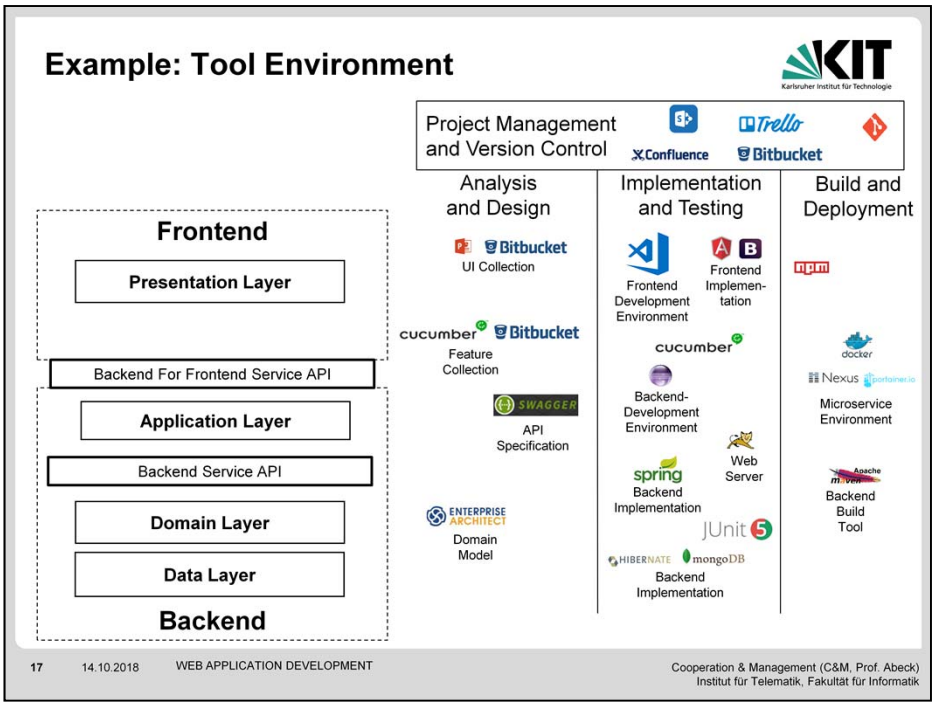
Software engineering tools cover two major aspects which must be handled in every software project: the management aspect dealing with organizational and project-related problems and the development aspect dealing with technical and (software) system-related problems.

(Communication and Collaboration Platform) The tools for solving either organizational or technical problems of software engineering are not decoupled. If a technical problem occurs this has influence on the overall planning. In order to manage these relationships a common communication and collaboration platform for management and development is useful.

(Software Repository) A software repository enables the storage of different versions of the software in a way that each software engineers can communicate the changes he has made in a systematic way to the other team members.

(Project Management and Organization) The relevance of this group of software engineering tools rises with the size of the projects and the software engineers involved.
(Project Planning and Tracking) This type of tool is used by a project manager (in Scrum this role is called project owner) to plan the resources (esp. the developers) that are available in the project. In [Ca01] the project planning and tracking is part of the so-called software engineering management tools.
(Issue and Problem Tracking) A team of software engineers uses this type of tool to organize technical, i.e. software-related tasks.

(Software Development) In this group of tools the development process starting with the analysis and ending with the deployment of the software is supported. The list of tool types mentioned in the figure is not complete. Tool types which are missing include the software quality tools or software maintenance tools.



The main dimension according which the development tools can be ordered are the development phases (from analysis to deployment). The tools supporting the analysis and design and the implementation and testing can be grouped according to the software architecture (which in our case is a microservice architecture).

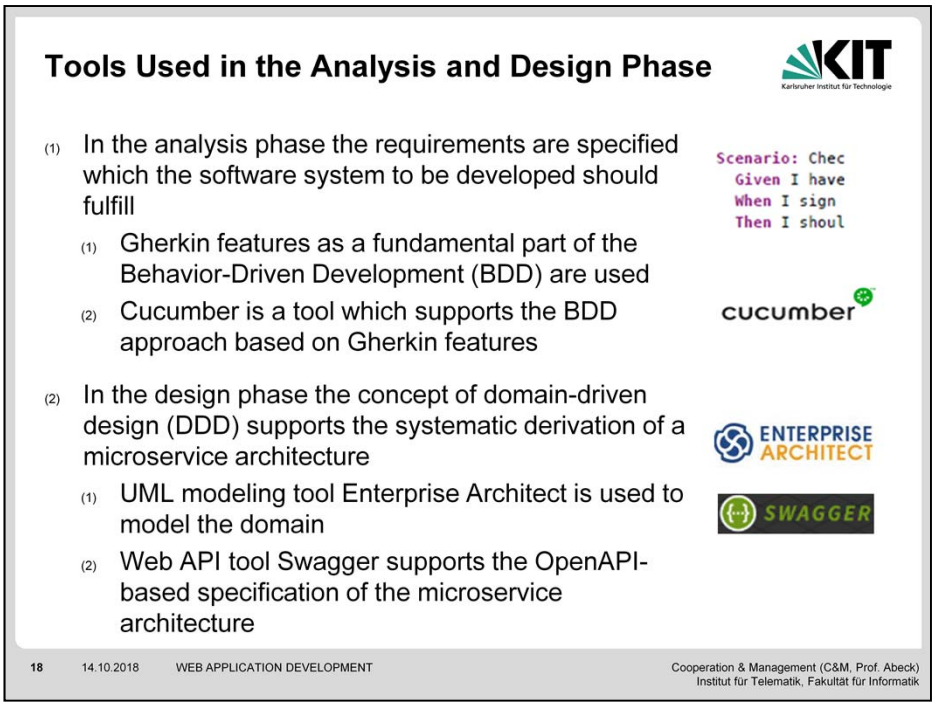
(Project Management and Version Control) These tools support the overall organization of the software project and the communication between the project members. At C&M two different tools sets are used: the Atlassian toolset extended by Microsoft tools, esp. SharePoint on which the C&M Teamserver is based.

(Analysis and Design) Analysis requirements at C&M is done by taking the approach of behavior-driven development (BDD) based on the tool Cucumber. For the design the two most relevant tools are Enterprise Architect for the domain model and Swagger for the API specification. In addition to these tools the Microsoft Office tools (Word, PowerPoint) and the Bitbucket Wiki are applied for documentation purposes.

(Implementation and Testing) In this phase the frontend and backend of the web applications are constructed. The integrated development environment (IDE) used for frontend development is Visual Studio Code and frontend frameworks are Angular and Bootstrap. As IDE for backend development Eclipse is used and Apache Spring (esp. Spring Boot for the microservice implementation) is used as backend framework.

(Build and Deployment) The build and deployment of the microservices is carried out via a build pipeline by which the concept of continuous integration and continuous deployment is provided. The result of the build pipeline is a Docker image (= application container image) since Docker is used as the container environment at C&M. Kubernetes is used to manage the Docker containers in order to reach a high scalability and robustness of the service landscape.

BDD	Behavior-Driven Develoment
IDE	Integrated Development Environment

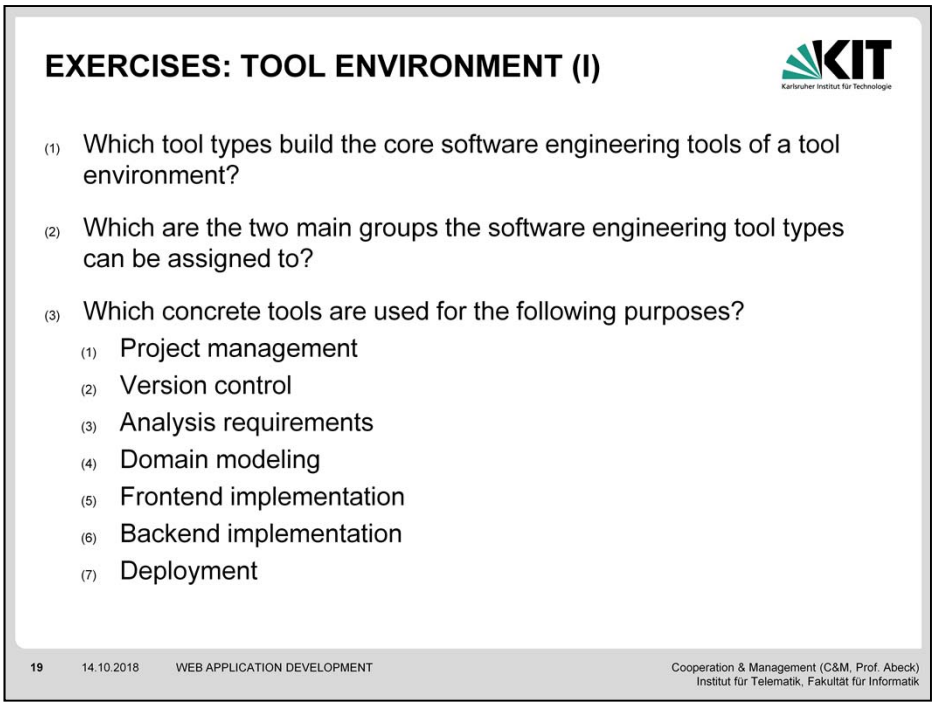


Analysis and design comprises the development work that is needed to implement the right software in the right way.

(1) Analysis is concerned with the question which software the user requires [CM-W-BEH].
(1.1) Gherkin is a language which introduces a minimal structure into the largely informal description of the requirement description.
(1.2) Cucumber allows to test an implemented software based on the specified features. The test are called user acceptance tests.

(2) Design is concerned with the static and dynamic structures of the software system that is to be implemented [CM-W-DOM].
(2.1) The concept of domain-driven design (DDD, [Ev04]) does not include a way how to specify the parts of a domain model. The Unified Modeling Language (UML) is a widely accepted modeling language which provides the flexibility to express the needed DDD modeling elements. Enterprise Architect is one of the leading professional UML tools to implement the DDD-to-UML approach.
(2.2) Based on a formalized UML-based domain model, the web application programming interfaces (API) of the microservice architecture can be systematically derived. The specification of the APIs is based on the OpenAPI defacto standard. Swagger is a well-known tool to specify and test the web APIs.

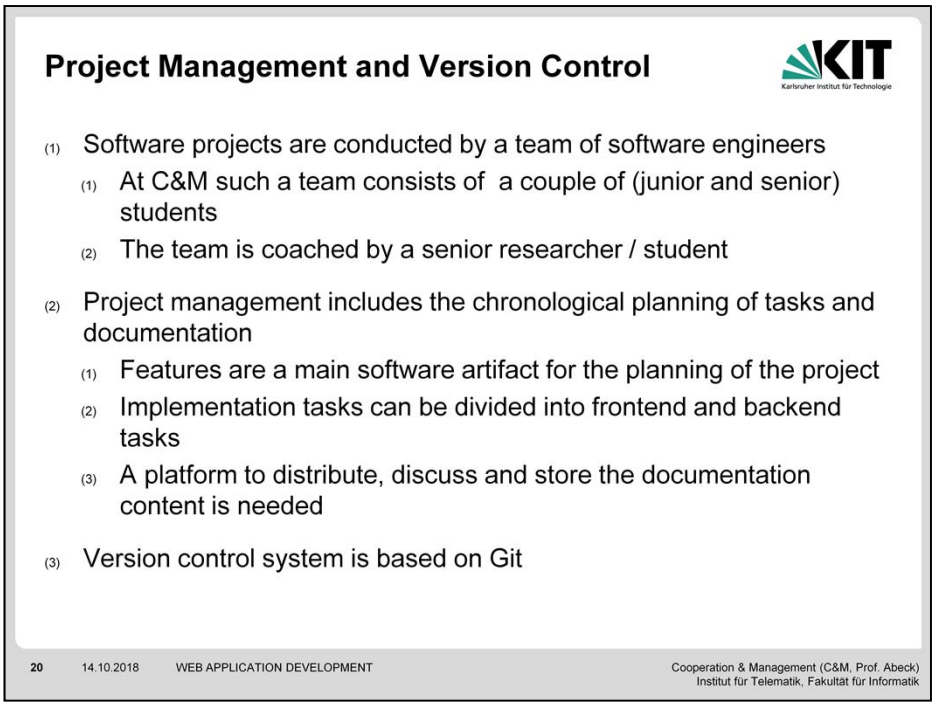
[CM-W-BEH] Cooperation & Management: BEHAVIOR-DRIVEN DEVELOPMENT, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*
[CM-W-DOM] Cooperation & Management: DOMAIN MODELING, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*
[Ev04] Eric Evans: Domain-Driven Design – Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.



(1) ++Spectrum and Types of Tools++
- Software construction tools: Program editors, compilers / interpreters, debuggers

(2) ++Classification of Software Engineering Tools++
- Project Management and Organization: The relevance of this group of software engineering tools rises with the size of the projects and the software engineers involved.
- Software Development: In this group of tools the development process starting with the analysis and ending with the deployment of the software is supported.

(3) ++Example: Tool Environment++
(3.1) SharePoint, Confluence, Jira, Bitbucket, Trello
(3.2) Git, Bitbucket
(3.3) Cucumber
(3.4) Enterprise Architect
(3.5) Angular and Bootstrap
(3.6) Spring
(3.7) Docker, Nexus, Portainer



In this chapter the software engineering tools are introduced that are used to manage and organize the project and the people involved in the whole software engineering process.

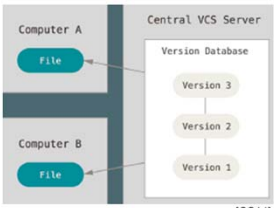
(1) Software systems tend to become complex which means that a whole team of software engineers is needed to develop the system. That is why software engineering means teamwork.
(1.1) The project teams are built in the beginning of each semester.
(1.2) The coach takes care that all team members contribute to the teamwork.

(2) In this context, project management comprises all tasks that should be carried out to organize the development tasks in a systematic and clear way.
(2.1) Features specify the requirements the software system must fulfill. They are described in a language called Gherkin. Since the agile development process is organized along the features with highest priority this artifact plays an important role for the project planning.
(2.2) The division into frontend-related and backend-related tasks make sense since they demand different competences.
(2.3) At C&M, the Bitbucket wiki is used to store the artifact descriptions which are edited collaboratively by the members of the development team.

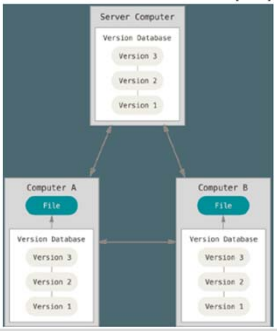
(3) The development of Git started in 2005. It is currently the most widely used version control system.

Version Control System (VCS)

(1) A VCS helps a developer team to manage changes to source code over time
  (1) Three types of VCS: Local, Central, Distributed
(2) Central VCS has a single server that contains all the versioned files
  (1) Examples: CVS (Concurrent Versions System), Subversion
(3) Distributed VCS fully mirrors the repository
  (1) Each system holds a full backup of all data
  (2) Examples: Git, BitKeeper, Mercurial



[CS14]



21 14.10.2018 WEB APPLICATION DEVELOPMENT Cooperation & Management (C&M, Prof. Abeck) Institut für Telematik, Fakultät für Informatik

(1) A version control system (VCS) is one of the most relevant software tools to develop complex software in a team [Atl-Wha].
(1.1) A local VCS is a database that keeps all the changes to files under revision control. The differences from one file version to the next file version usually is stored in a certain format. An example of a local VCS is RCS (Revision Control System). The central and distributed VCS are explained in what follows [CS14].

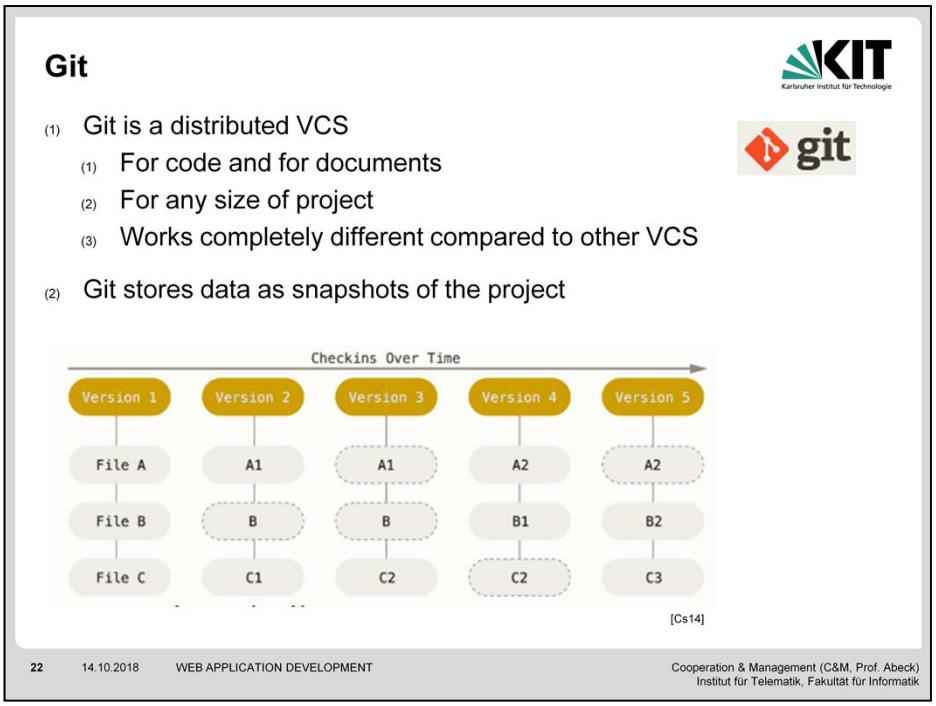
(2) A disadvantage of local VCS is that a collaboration of developers working on distributed systems is not supported which has led to the central VCS. In a central VCS, clients running on different systems check out files from the central place.
(2.1) Subversion is the successor of CVS which stands for Concurrent Versions System.

(3) In a central VCS, clients only check out single files and not the repository (i.e. the whole file system).
(3.1) The full mirror includes the full history of all the files in the repository. This leads to a high stability.
(3.2) BitKeeper today is an open-source distributed VCS which was used to develop the Linux kernel. Some conflicts caused by the fact that BitKeeper at that time was proprietary (owned by a company called BitMover) has led to the development of Git.
Mercurial is a free software which was announced and developed at the same time as Git in 2005.

(Computer A <---> Computer B) Note: There is only an "indirect" relationship between the participating computers A and B (leading via the server computer).

CVS	Concurrent Versions System
RCS	Revision Control System
VCS	Version Control System

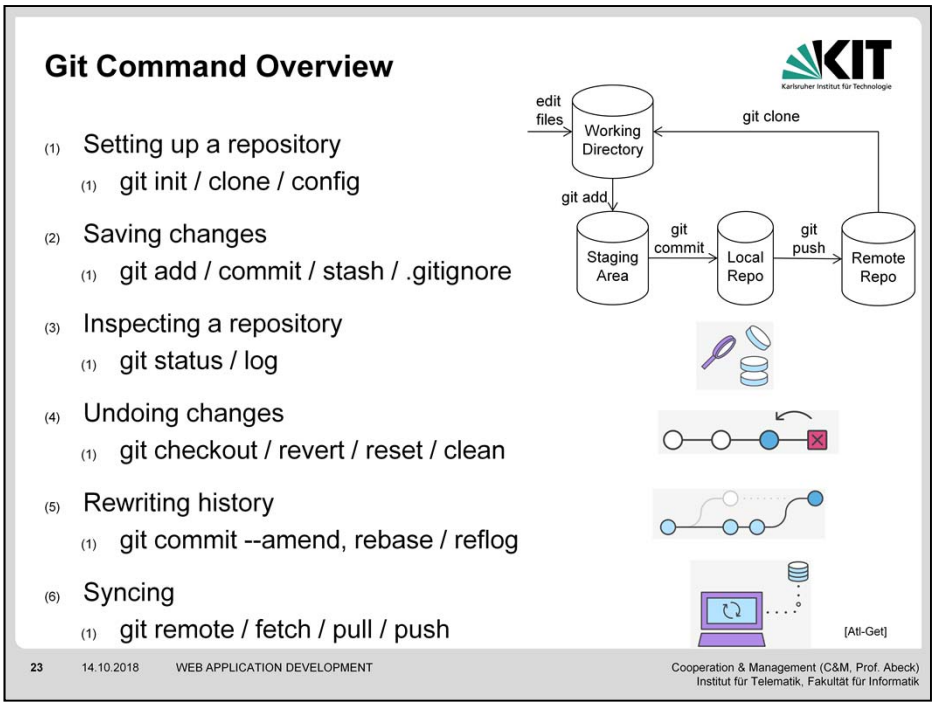
[Atl-Wha] Atlassian: What is version control. *https://www.atlassian.com/git/tutorials/what-is-version-control*
[CS14] Scott Chancon, Ben Straub: Pro Git, Apress, Second Edition, 2014. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*



(1) Git was developed by Linus Torvalds, the initiator of the Linux kernel. Development started in 2005 [HP16]. The reason for the development of Git was the need of a version control system for the Linux kernel development.
(1.1) Git allows to record and manage the changes of files.
(1.2) Git is the appropriate tool both for single persons and for large projects in which hundreds of developers cooperate (such as the Linux kernel development).
(1.3) Examples of other VCS include CVS (Concurrent Versions System) or Subversion (successor of CVS) which store data as a list of file-based changes (since the changes of files are stored as deltas, they are named delta-based version control).

(2) A snapshot in Git is a picture of all the files of the project. For efficiency reasons for all files that did not change from version to the other only a link to the previous identical file is stored.
(Version 3, A1, B, C2) The snapshot Git takes for Version 3 are links for A1 and B since they did not change (illustrated by the dotted lined ellipses). Only C2 is stored since it is different from C1 (which is file C in Version 2).

[Atl-Bec] Atlassian: Become a git guru. *https://www.atlassian.com/git/tutorials*
[CS14] Scott Chancon, Ben Straub: Pro Git, Apress, Second Edition, 2014. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*
[HP16] Valentin Haenel, Julius Plenz: Git - Verteilte Versionskontrolle für Code und Dokumente, 2016. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*



This page gives an overview of the most relevant commands provided by the distributed version control system Git [Atl-Get] .

(1) A Git repository (also called Git repo) is a virtual storage of the versions of code from a project.
(1.1) git init creates a new .git subdirectory in the current working directory. It will also create a new master branch.
git clone <repo url> creates a copy of the indicated remote repository. The URL format depends on the network protocol that is used.
git config sets Git configuration values (e.g. email or username) on a global or local project level. It also allows to create shortcuts for frequently used Git operations in order to work with Git more efficiently.

(2) In Git, each change of the local repository is done via a so-called staging area which can be seen as a buffer between the working directory and the project history.
(2.1) git add and git commit compose the fundamental Git workflow. They are the means to record versions of a project into the repository's history. The stage is used by Git to group related changes into highly focused snapshots before actually committing it to the project history (when git commit is executed).
git stash (germ. bunkern) temporarily saves uncommitted changes and reverts them when a developer later wants to continue his work.
.gitignore is a file which contains all names of files that should be ignored by Git (e.g. build artifacts or machine generated files, such as compiled code).

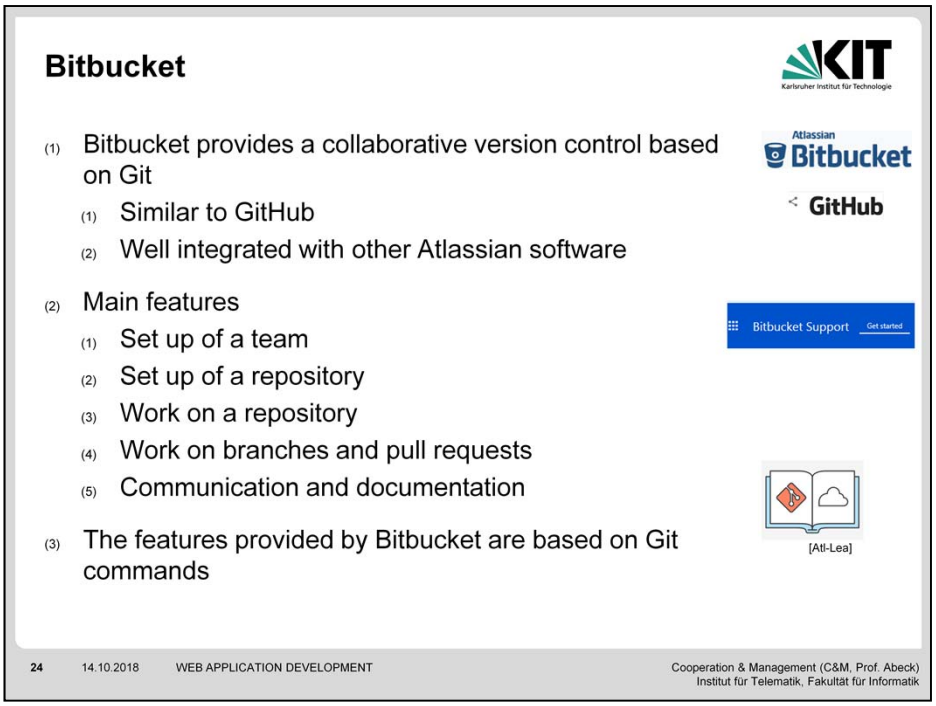
(3.1) git status displays the state of the working directory and the staging area.
git log provides information regarding the committed project history.

(4) These commands allow to explore old commits and undo the changes in different ways.

(5) By using these commands the Git history can be rewritten or altered.

(6) These commands are used by a developer to share a series of commits with other developers.

[Atl-Get] Atlassian: Getting Started. *https://www.atlassian.com/git/tutorials/setting-up-a-repository*

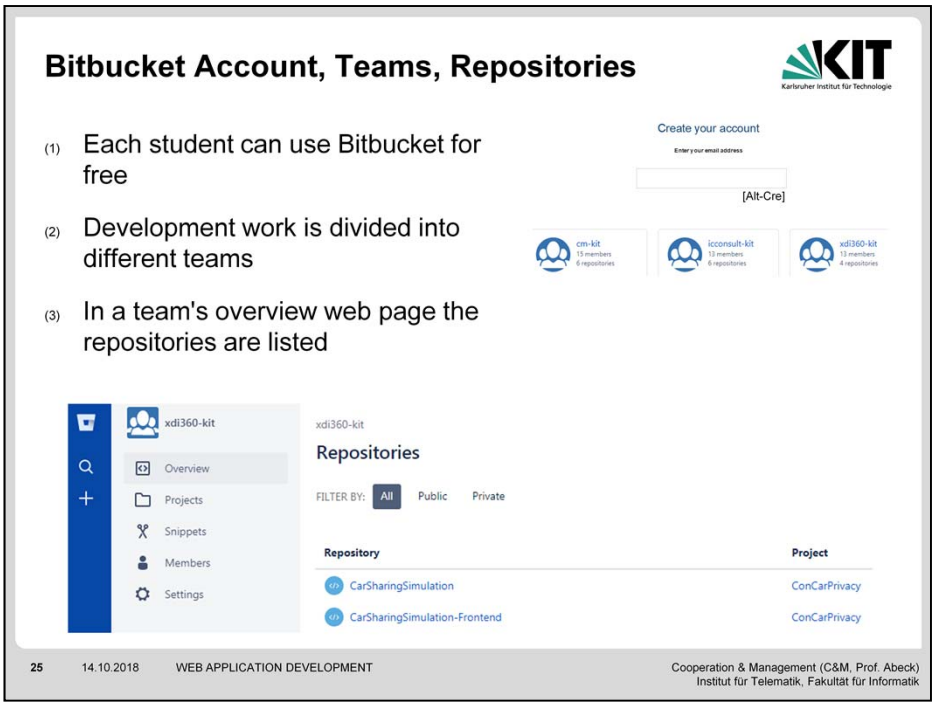


Bitbucket owned by Atlassian provides a web-based hosting service. It is written in Python using the Django web framework.

(1) The first version of Bitbucket (developed in 2010 by an independent startup) only offered hosting support for Mercurial projects.
(1.1) In contrast to other open source hosters, such as SourceForge, in GitHub not the project as a collection of source code is in the center, but the user and his repositories which are managed by Git. Operations, such as branching (e.g. creating a new repository) and merging of forked repositories are supported (https://de.wikipedia.org/wiki/GitHub).
(1.2) The integration of Bitbucket with other Atlassian software, such as Confluence (communication), Jira (incident management) and Bamboo (continuous integration and continuous deployment) leads to complete development solution.

The main features provided by Bitbucket allow a team to efficiently apply the Git-based version control concepts. The following description is based on the "Bitbucket Support" documentation provided by Atlassian [Atl-Bit].
(2.1) This feature includes all operations to create and change a team and set permissions for the team members.
(2.2) The set up includes the operations of creation, import, and cloning of repositories.
(2.3) The work on a repository includes operations to push code from the local repository to Bitbucket or to pull code from Bitbucket to get the most up-to-date version onto the local repository.
(2.4) A developer who wants to work on a separate line of code first creates a new branch or checks out an existing branch. When the code is ready to be reviewed a pull request is created. After the review the branch is merged into the main branch.
(2.5) The communication takes place via issues by which tasks and bug reports can be exchanged. By the creation and editing of wiki pages everything related to the project, team, and code can be documented.

[Atl-Get] Atlassian: Get started with Bitbucket Cloud. *https://confluence.atlassian.com/get-started-with-bitbucket/get-started-with-bitbucket-cloud-856845168.html*
[Atl-Lea] Atlassian: Learn Git with Bitbucket Cloud. *https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud*

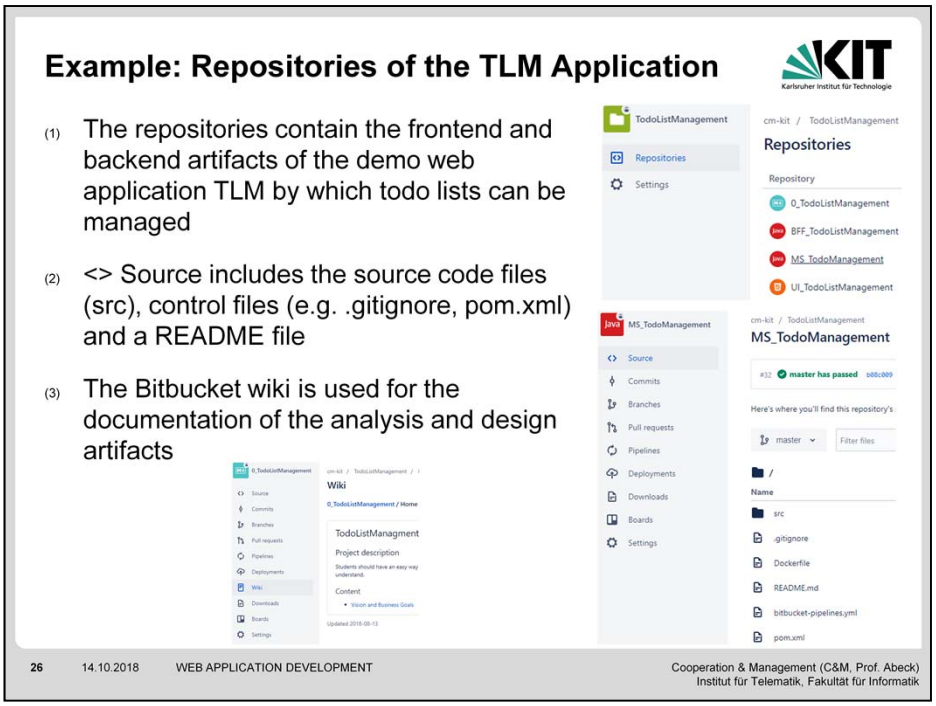


(1) C&M has got a free academic licence to use Bitbucket in its research projects. Students can get a Bitbucket account via [Atl-Cre]. It is important to use the KIT mail address. If a student already has a Bitbucket account based on his/her private mail address this account should not be used in C&M projects. The licence-related aspects are further described in [CM-C-Dur]. A confluence account offered by iCC/xdi does not include a Bitbucket account.

(2) The Bitbucket repositories are owned by teams. For each cooperation partner a team with different repositories exists.
Each C&M member should be part of one or more teams (click on profile icon | View profile | Teams).

(3) The example shows the Bitbucket repositories used for the software that is being developed in cooperation with xdi360. In the Bitbucket documentation, repositories and their use are explained in detail.

[Atl-Cre] Atlassian: Create Your Account. *https://bitbucket.org/account/signup/*
[Atl-Rep] Atlassian: Repositories. *https://confluence.atlassian.com/bitbucket/repositories-675385631.html*
[CM-C-Dur] Cooperation & Management: Durchführung von Entwicklungsarbeiten, Confluence-Seiten.
https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/CM-Dokumente
https://ic-consult.atlassian.net/wiki/spaces/KK/pages/121602185/Durchf+hrung+von+Entwicklungsarbeiten



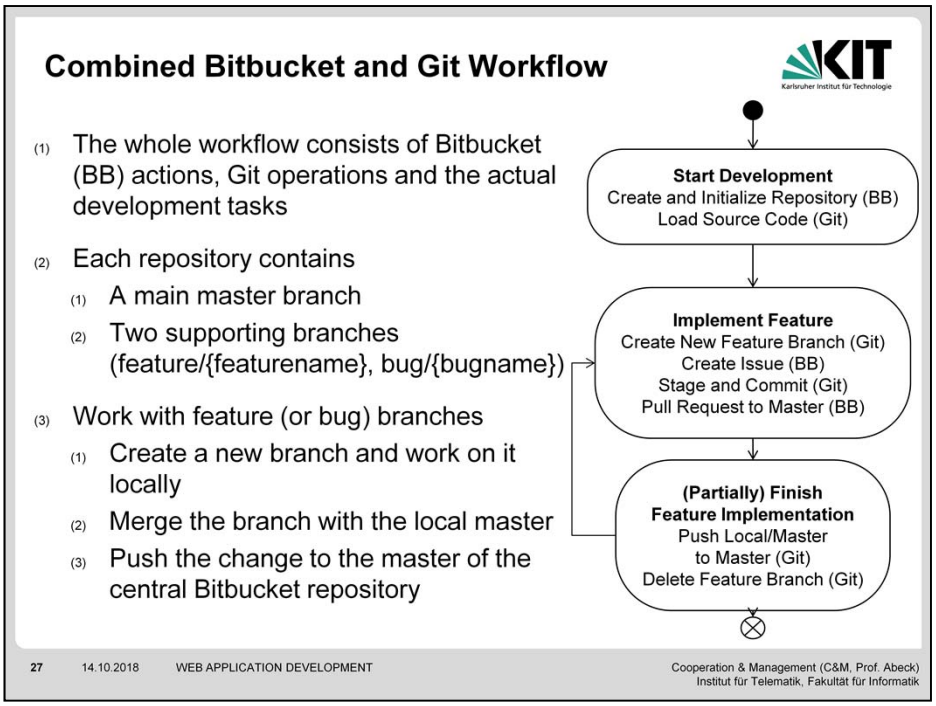
The structure and content of a Bitbucket repository is shown with the repositories of the TLM web application.

(1) The TLM demo provided by C&M is accessible via Bitbucket [CM-B-TLM].
(cm-kit / TodoListManagement) The repositories assigned to the team "cm-kit" and the project "TodoListManagment" being one of the cm-kit's project.
(0_TodoListManagement) For each project a repository "0_<project name>" should exist in which the artifacts that are valid for the whole project (i.e. frontend and backend) are stored.
(MS-TodoListManagement) This is the repository in which the backend microservices (MS stands for MicroService) of the TLM application are stored.
(UI-TodoListManagement) This is the repository in which the frontend code (UI stands for UserInterface) of the TLM application are stored.

(2) All the information stored in the source folder is loaded into the development environment by using the appropriate git command (which is "git clone https://bitbucket.org/cm-tm/ms-todolistmanagement.git" in the case that the backend source code is loaded).
(master has passed) Pipeline and commit status information
(master, Filter files) By these fields the source can be browsed along the branches and filtered according to the file names.
(src) The source code files are part of the folder "src". An example of a source code file which describes the todo domain object is: "src/main/java/edu/kit/tm/cm/springdemo/domain/model/Todo.java".
(gitignore, Dokerfile, bitbucket-pipelines.yml, cm.xml, pom.xml) These are the control files needed for building and deploying the software.
(README.md) In this file the most relevant information can be found which describes what the software provides, how it can be used, and what the current status and open tasks are.

(3) The wiki provided by Bitbucket allows to carry out the documentation of the software in a collaborative way since each team member can make contributions to the wiki pages.

[CM-B-TLM] Cooperation & Management: Bitbucket project TLM. *https://bitbucket.org/account/user/cm-tm/projects/TLM*



The workflow described on this page is inspired by the Git workflow described in [Atl-Git]. A more detailed description of the workflow outlined on this page can be found at [CM-C-Dur].

(1) In the graphical presentation of the workflow the actual development tasks are left out. Examples of such tasks are the development of the software implementing the feature or the solution of an issue (e.g. fixing of a bug).

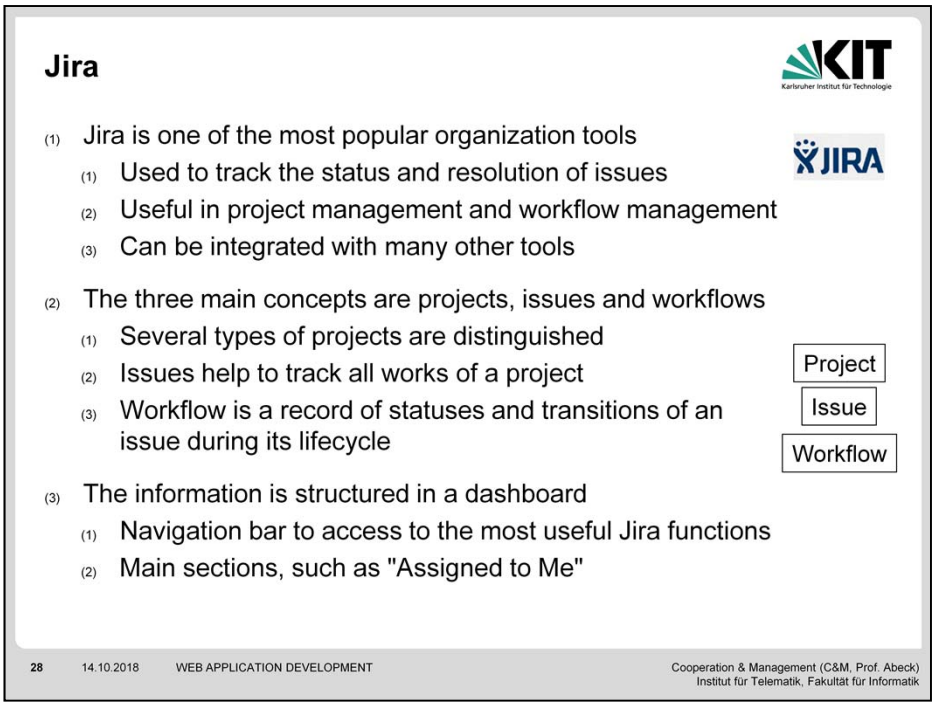
(Create and Initialize Repository (BB)) The precise Bitbucket commands to create a new repository can be found at [Atl-Cre].

(2) A branch represents an independent line of development for the repository. A branch includes a new working directory, staging area, and project history. Before any new branches can be created, the main branch called master is automatically created.
(2.1) A second main branch containing the last stable version which was deployed can exist.
(2.2) In the feature branch new functionality is developed whereas in the bug branch faults in the software are solved. There is no difference how to work with these branches by using Bitbucket and Git operations.

(3) The (feature-related or bug-related) changes on the software are carried out by the following three steps: (i) locally create and work on a new branch, (ii) merge the changed branch into the local master branch and (iii) push the change to Bitbucket. Theses steps are described in the Bitbucket documentation [Atl-Use].
(3.1) The terminal window should be opened and the current folder should be the top level of the repository: cd ~/repos/<repo-name>
A local branch can be created by using the following git command: git branch <feature/featurename>
To begin working on the new branch, it must be checked out: git checkout <feature/featurename>
Stage the changed file: git add <filename>
Commit the changed file: git commit <filename> -m <comment explaining the change that is committed>
(3.2) Again, the terminal window should be opened and the current folder should be the top level of the repository: cd ~/repos/<repo-name>
It should be made sure that the feature branch to be merged is checked out and all changes have been committed. This information is provided by: git status
The master branch is to be checked out: git checkout master
Now, the changed feature branch can be merged to the master branch: git merge <feature/featurename>
(3.3) In order to make the changes available to the other developers of the team the state of the local repository (Local/Master) must be pushed to the central Bitbucket repository (Origin/Master): git push origin master

BB Bitbucket

[Atl-Cre] Atlassian: Create a Git repository. *https://confluence.atlassian.com/bitbucket/create-a-git-repository-759857290.html*
[Atl-Git] Atlassian: Gitflow Workflow, Git Tutorial. *https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow*
[Atl-Use] Atlassian: Use a Git branch to merge a file. *https://confluence.atlassian.com/bitbucket/use-a-git-branch-to-merge-a-file-681902555.html*
[CM-C-Dur] Cooperation & Management: Durchführung von Entwicklungsarbeiten, Confluence-Seiten.
https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/CM-Dokumente
https://ic-consult.atlassian.net/wiki/spaces/KK/pages/121602185/Durchf+hrung+von+Entwicklungsarbeiten



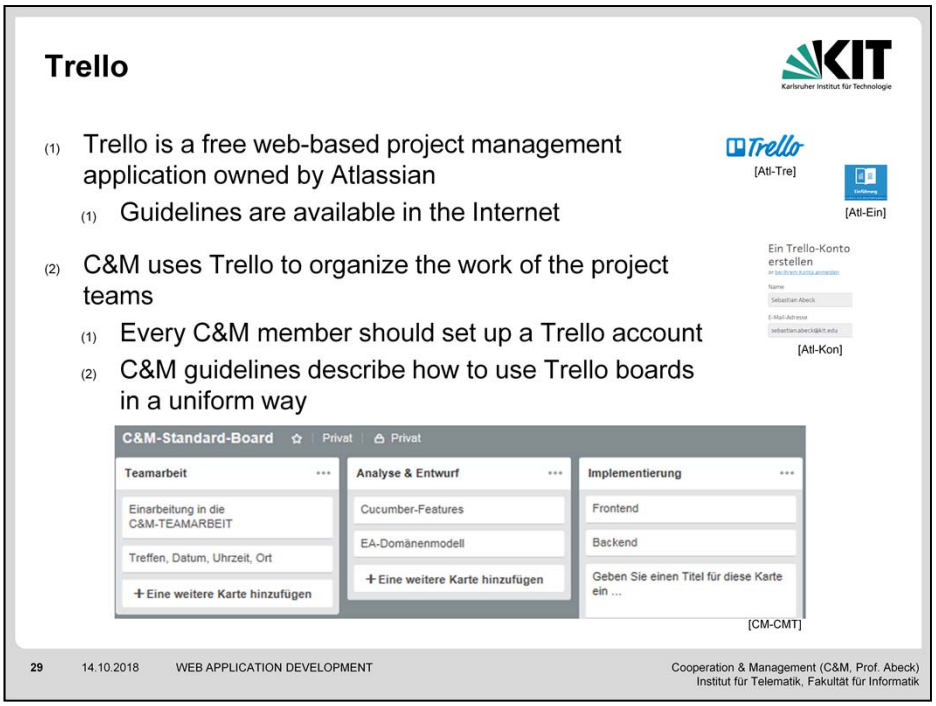
The name Jira stems from the Japanese name for Godzilla (the name of a monster) which itself is a reference to Jira's main competitor, Bugzilla (an open source software released by Netscape Communications). It is pronounced "Jee-rah". The software is developed by Atlassian since 2002 and it is written in Java [Wik-Jir].

(1) Jira is used for Project Management, Bug Tracking, Issue Tracking and Workflow [tut-Jir].
(1.1) This functionality is needed in help desk, support and customer services.
(1.2) This includes task tracking, requirement management and process management.
(1.3) Such tools include source control programs such as Git or Version Control Systems (CVS).

(2) Issues are the central concept since a project is a collection of issues and a workflow tracks the lifecycle of an issue.
(2.1) Examples for projects are: software development project, marketing project, employee performance system.
(2.2) Examples of issues are: task or sub-task of a story, bug or defect, helpdesk ticket can be logged as an issue.
(2.3) Examples of statuses an issue are: open, in progress, resolved, reopened, close

(3) The Jira dashboard is the first page that shows up (*https://www.tutorialspoint.com/jira/jira_dashboard.htm*).
(3.1) These links are projects, issues, boards and create.
(3.2) In this section the issues list assigned to users is displayed. Another example of a main section is "Activity Stream" which display activities done by the user.

[tut-Jir] tutorialspoint: Jira. *https://www.tutorialspoint.com/jira/*
[Wik-Jir] Wikipedia: Jira. *https://en.wikipedia.org/wiki/Jira_(software)*

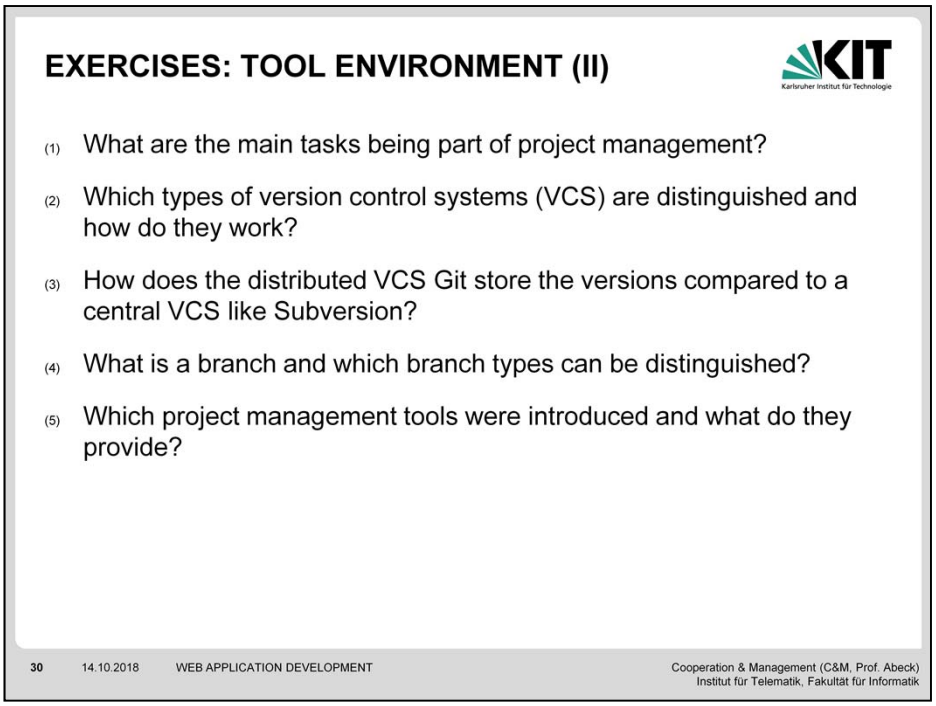


Trello is a lightweight alternative for professional project management tools like Jira.

(1) Originally, Trello was developed by Fog Creek Software in 2011 and sold by Atlassian in January 2017.
(1.1) There is extensive information available ([Atl-Ein], in German) for what purposes and in which way Trello can be used.

(2) The functionality of Trello is used by C&M to establish a Scrum board which provides a structured way to organize the development tasks of all members of the project team.
(2.1) It is recommended to use the KIT mail address when signing up the Trello account.
(2.2) The guidelines include a common list structure of each Trello board. A detailed description is part of the C&M teamwork [CM-CMT].

[Atl-Ein] Atlassian: (Trello-) Einführung. *https://trello.com/guide/getting-started.html?utm_source=welcome&utm_medium=email&utm_campaign=Welcome*
[Atl-Kon] Atlassian: (Ein Trello-) Konto erstellen. *https://trello.com/signup*
[Atl-Tre] Atlassian: Trello. *https://trello.com/*
[CM-CMT] Cooperation & Management: C&M-TEAMARBEIT, Teamarbeitsdokument. *https://team.kit.edu/sites/cm-tm/Mitglieder/1-1.Teamarbeit*



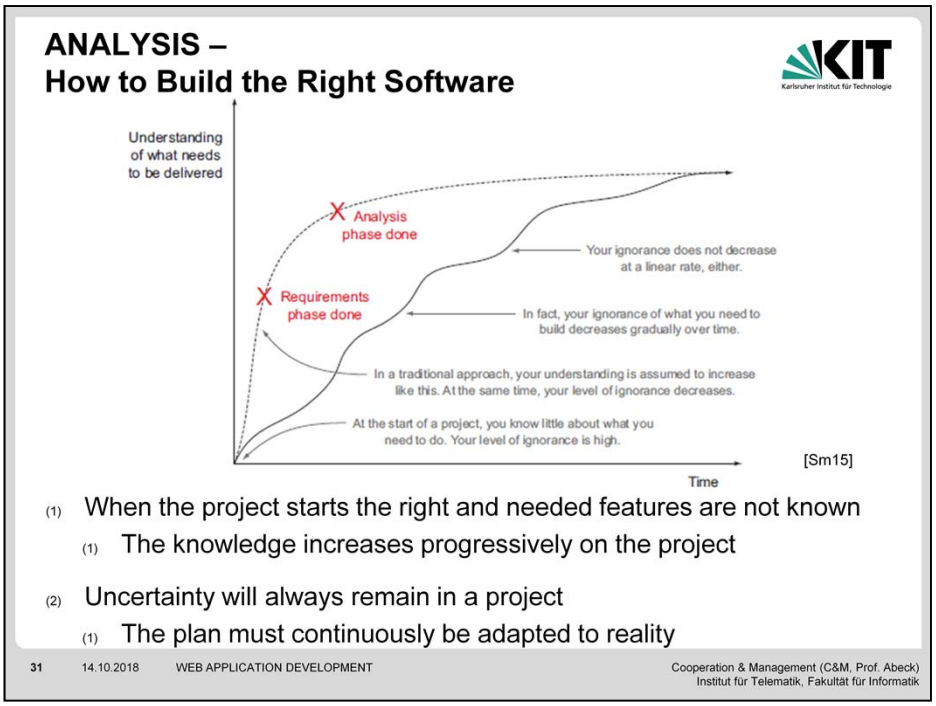
(1) ++Overview++
- Chronological planning
- Distribution of tasks
- Documentation

(2) ++Version Control System++
- Local: VCS Client and VCS server run on one system. Disadvantage: collaboration of developers working on distributed systems is not supported.
- Central: Clients running on different systems check out files from the central place.
- Distributed: Client fully mirrors the repository. Each system holds a full backup of all data.

(3) ++Git++
- Git stores snapshots in each client. A snapshot is a picture of all the files of the project. For efficiency reasons for all files that did not change from version to the other only a link to the previous identical file is stored.
- Subversion (successor of the Concurrent Versions System CVS) stores data centrally on the server as a list of file-based changes (since the changes of files are stored as deltas, they are named delta-based version control).

(4) ++Bitbucket / Git Workflow @ C&M++
- Branch: represents an independent line of development for the repository. A branch includes a new working directory, staging area, and project history.
- Types
-- Master (origin/master): Before any new branches can be created, the main branch called master is automatically created.
-- Stable (origin/stable): This branch is optional and contains the last stable version which was deployed.
-- Feature (feature/featurename): In this branch new functionality is developed
-- Bug (bug/bugname) In this branch faults in the software are solved. There is not difference how to work with these branches by using Bitbucket and Git operations.

(5) ++Jira++ ++Trello++
- Jira: Professional incident management tool which allows to track incidents based on flexible workflows
- Trello: A simple project management tool which provides boards on which a team can organize its tasks

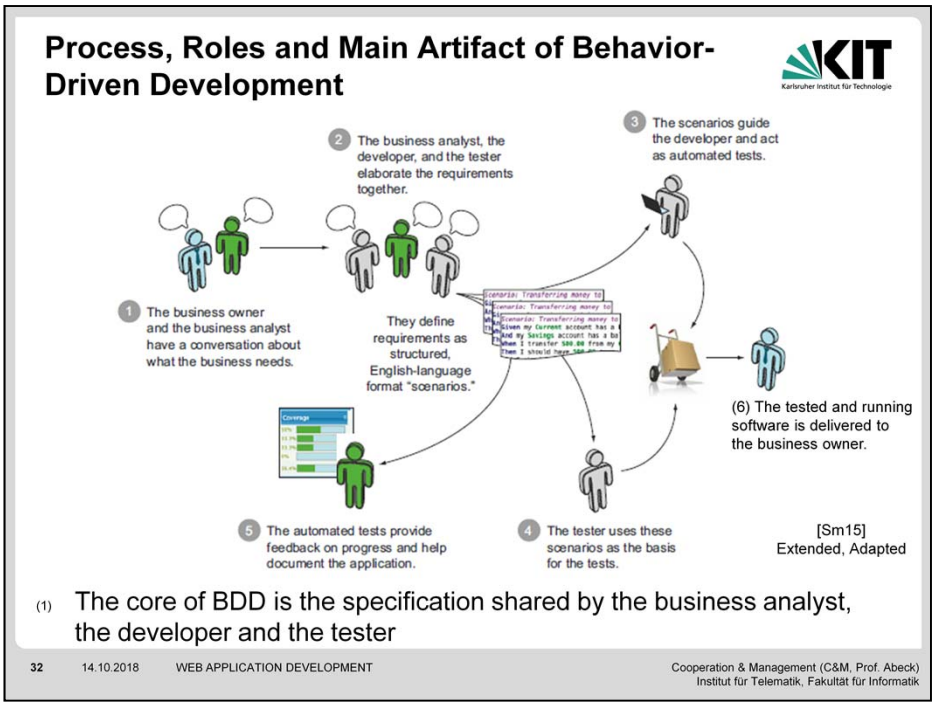


The right software can only be built when the problem the software should solve is well understood.

(1) At project start the level of ignorance is at its maximum.
(1.1) The learning path is neither linear nor predictable.

(2) It is hard to predict what the development team will learn as the project progresses.
(2.1) It is not advisable trying to force reality to fit into your plan according to a Swiss Army proverb: "When the terrain disagrees with the map, trust the terrain."

[Sm15] John Ferguson Smart: BDD in Action – Behavior-Driven Development for the whole software lifecycle. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD*



A main characteristic of the process behind the Behavior-Driven Development (BDD) is the collaboration of three central roles (business analyst, developer, tester) in the development process. A fourth role is the business owner using the software.

(1 The business owner ... business needs.) The first step is similar to the traditional development process.

(2 The business analyst ... requirements together.) Two aspects in BDD are new and different compared to the traditional process:
(i) The collaboration of the business analyst, developer and tester.
(ii) The structured specification all three roles agree on.

(3 The scenarios guide the developer ...) For the developer the specification defines what should be implemented.

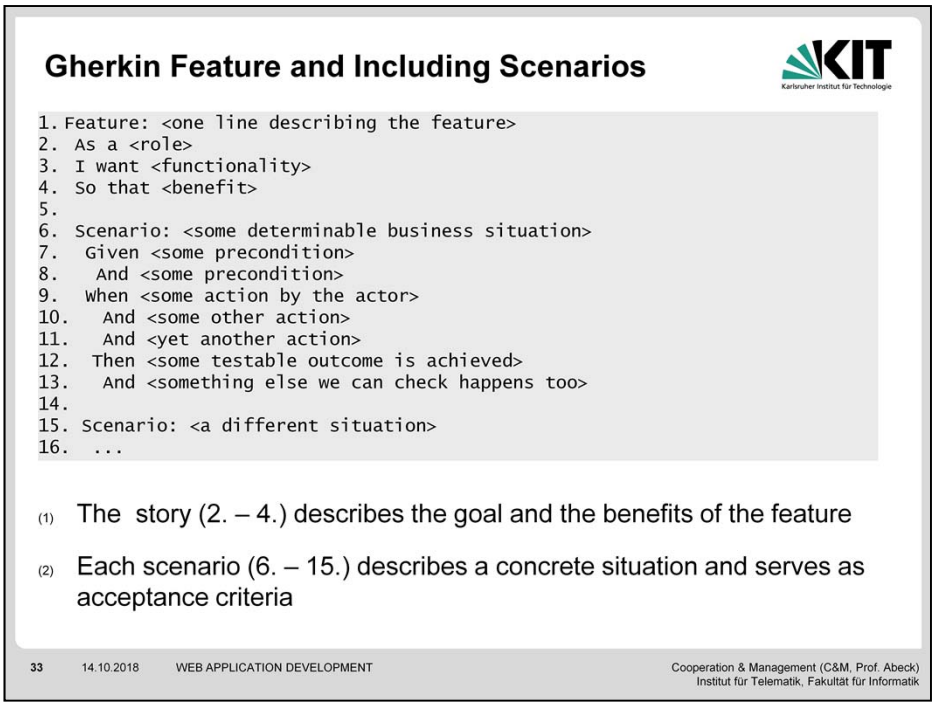
(4 The tester ... for the tests) For the tester the specification defines what should be tested.

(5 The automated tests ... document the application) For the business analyst (and the two other roles) the specification and the outcome of the tests document the software and its current state of implementation. The scenarios can be seen as a low-level technical documentation and provide up-to-date examples of how the system works.

(6) The structured collaboration of the roles taking part in the development process leads to a software which does the right thing and which does the thing right.

(1) The advantage of BDD compared to the traditional development process is the common specification which prevents misunderstanding and miscommunication, especially between the business analyst and the developer. This specification which is structured according to a defined grammar and keywords is the core of BDD. It consists of a so-called story and scenarios described in natural language.

(Extended, Adapted) The text "6 The tested ..." was extended. The role of the business analyst (i.e., a green instead of a grey stickman) was adapted in step 5.

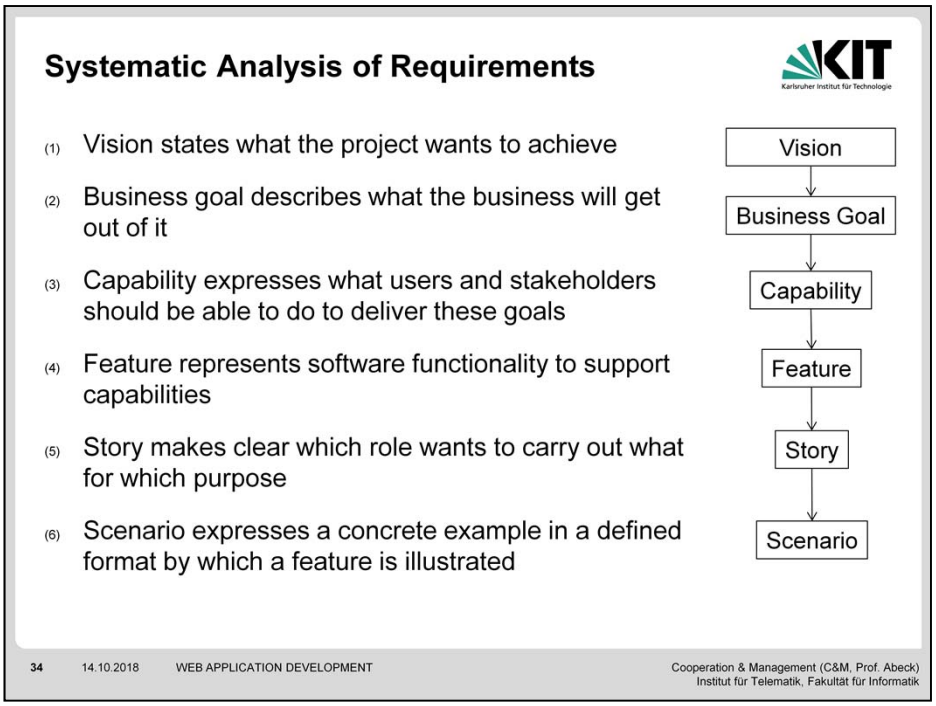


BDD provides a language Gherkin consisting of specific keywords used to define the features (and the scenarios to test these features) which the software system must fulfill. The proposed template is taken from two sources: The upper feature description part is taken from [DNA-Story] and the lower scenario description part is taken from [WH12].

(1) In the description the business value of the feature should be recognizable.
Although Gherkin does not prescribe any structure and use of specific keyword in this part, it is recommended to use the structure "As a – I want – So that" proposed by Dan North.

(2) Behind BDD's Given-When-Then concept a finite state machine can be seen.
The structure and the keywords "Given – When – Then" are prescribed by Cucumber.

[DNA-Story] Dan North & Associates: What's in a Story. *https://dannorth.net/whats-in-a-story/*
[WH12] Matt Wynne, Aslak Hellesoy: The Cucumber Book. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD*



Developers use lots of different words to describe what they want to build. The following terminology taken from [Sm15:88] is used to provide a clear understanding in this confusing part of software development.

(1) The project vision provides a high-level guiding direction for the project. The problem must be clarified to understand which software system is needed by which users to solve the problem .

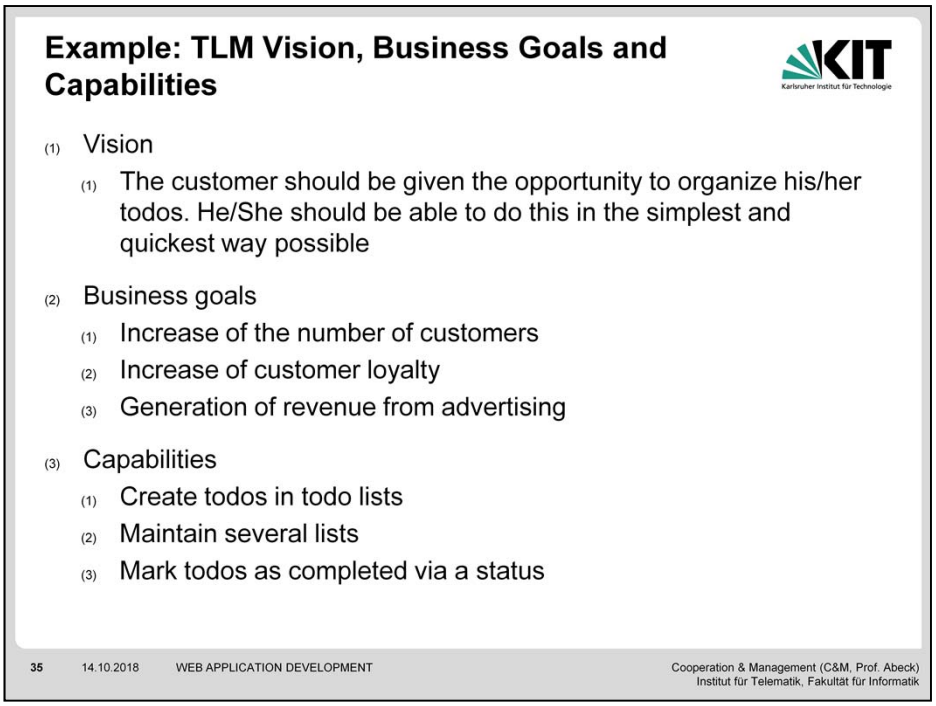
(2) The software system must have a measureable, positive impact on the business of the customer the software is built for.

(3) Capabilities give users or stakeholders the ability to realize some business goal or perform some useful task. A capability represents the ability to do something; it does not depend on a particular implementation. For example, "the ability to book a flight" is a capability.

(4) A feature is a piece of functionality that is delivered to the end users or to other stakeholders to support a capability that they need in order to achieve their business goals [:98].
A feature is something that users can test and use in isolation. A feature can deliver business value in itself; once a feature is completed, it could theoretically be deploy into production immediately, without having to wait for any other features to be finished first. The description of the features is generally more effective than describing what the application does.

(5) The typical format of a story is "As a – I want – So that".

(6) Scenarios written in the format "Given – When – Then" make up the core of an executable specification.



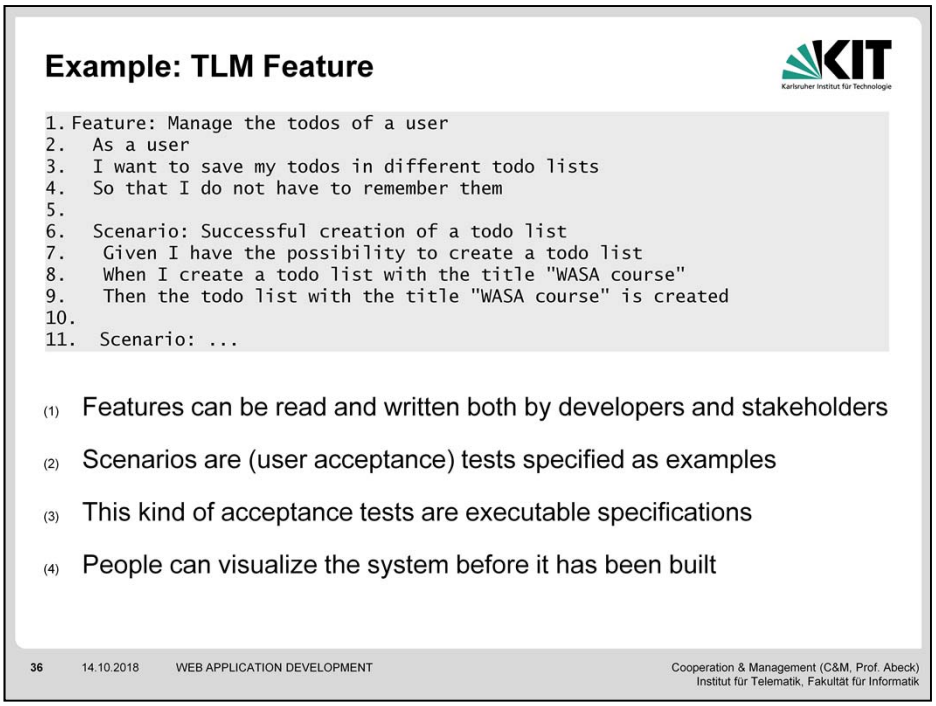
By defining the vision, business goals and capabilities, the software system to be developed can be described on a high abstraction level.

(1) The vision should make clear for which purpose the software system is used. In this case the minimal core functionality of the todo list management is to organize the todos in the easiest and quickest way possible. By this vision a Minimum Viable Product (MVP) is described.

(2.1) More customers are attracted by a modern application which is easy to use and which provides intelligent functionality.
(2.2) Offering such an application, customer loyalty will increase and additional products can be sold to the customer.
(2.3) Revenue can also be generated from the advertising shown in the application.
Remark: The business goal pursued by C&M is to practically demonstrate the students the BDD/DDD-based software development with an easy example.

(3) The formulation of the capabilities in this example is close to the features that the (minimal) todo list management system should provide.

MVP Minimum Viable Product



The example is taken from the TodoListManagement (TLM) which allows a user to organize todos in different lists.

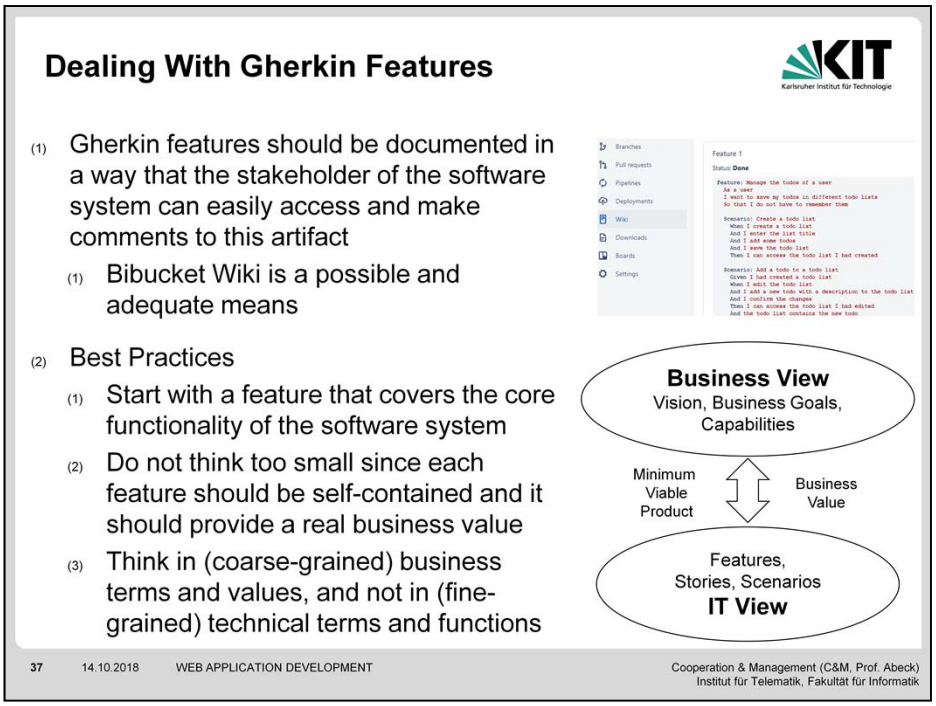
(1) As the example shows (user acceptance) tests are written in a way that they can be understood by everyone, especially also by non-technical experts who know the domain and users who should use the system.
(1. Feature: ...) The feature describes the functionality which concerns the actions a user can take in order to manage todos. All terms used in the description (e.g. todo , todo list, list title) are taken from the domain. Therefore, these terms are understood not only by the developer but also by the stakeholder.

(2) Each scenario describes one example case.
(6. Scenario: Successful creation of a todo list) In the example the scenario describes the case that all accessibility information available for a point of interest should be shown. A Further scenarios for this feature exist and describe examples which test the filtering, sorting, or pagination of the information.

(3) This is the reason for Dan North to use the term Behavior-Driven Design instead of Test-Driven Design since the specification of the behavior is the goal and the test is only a means to reach this goal.

(4) Since everyone can read and write a test like this a common understanding between stakeholder and developer of what the system should do can grow. Stakeholders are motivated to think about and write down further scenarios that are to be considered.
(7. Given I have the possibility to create a todo list) The steps should be formulated in a way that no assumptions are made with respect to the GUI (Graphical User Interface). Visualization can be further elaborated by additional UI prototypes illustrating the interaction of the user with the software.

GUI Graphical User Interface

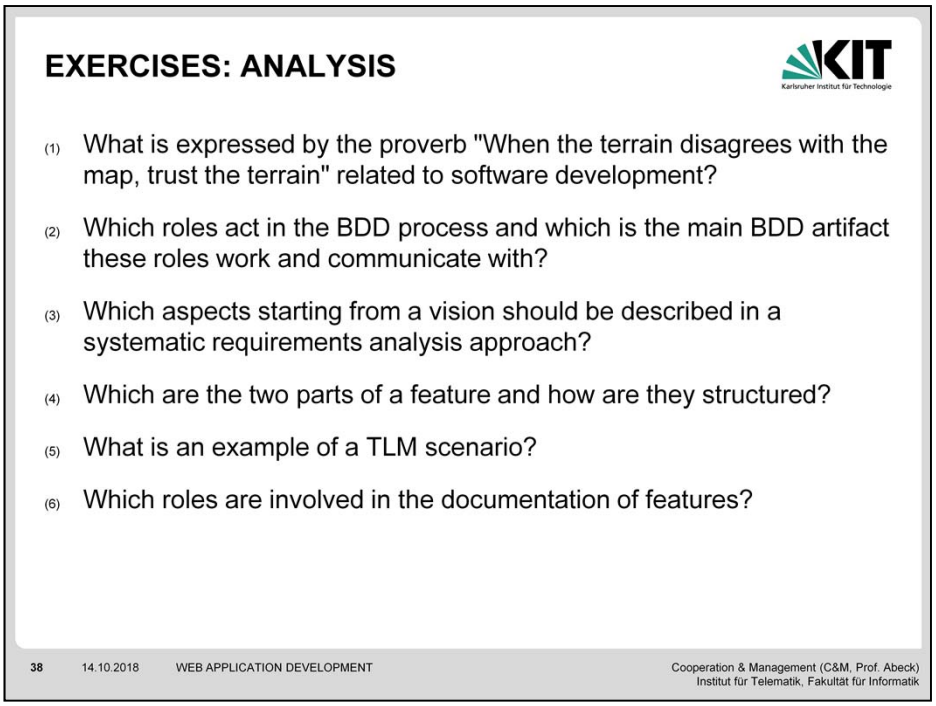


The artifact which describes the Gherkin features according to the BDD is the main outcome of the requirements analysis. Since all further development steps, esp. the modeling of the domain, depend on this artifact it should be documented in an adequate and defined way.

(1) The Gherkin features provide a requirements specification which is an integral artifact of the software development process. Therefore, developers must have access to the features in their IDE (Integrated Development Environment) since they run through the features in order to carry out user acceptance tests of their software implementation. Nevertheless, features are not only relevant for the developers, but also for the stakeholders for which the software is developed. Therefore, an adequate way of documentation of the feature collection must be made available not only for the developers, but also for the stakeholders.
(1.1) The

(2) The specification of features is a creative process which makes it impossible to state complete rules how to go through this process. Nevertheless, best practices can be formulated which give some valuable hints for the process.
(2.1) The first (one to three) features should specify the functionality of a so-called Minimum Viable Product (MVP).
(2.2) A business value will usually not be provided by one single functionality but by a set of coherent functions by which a user can fulfill a specific task.
(2.3) The terms of the ubiquitous language should be used. These terms should be understood by the software developer and the business owner/analyst.
Another best practice is to use a declarative and not an imperative way to express the scenario steps.

MVP Minimum Viable Product



(1) ++How to Build the Right Software++
The proverb concerns the analysis of the requirements the software should fulfill. It means that the plan must always be adapted to reality – and not the other way round.

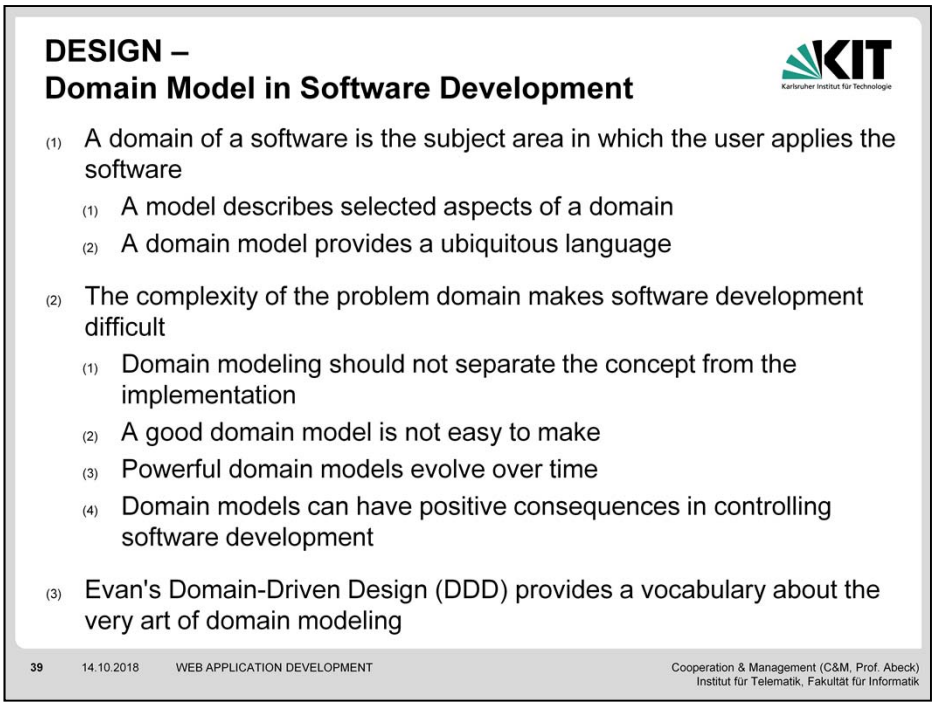
(2) ++Behavior-Driven Development (BDD) Process, Roles and Main Artifact++
- Three central roles: business analyst, developer, tester
- Scenario description: This specification which is structured according to a defined grammar and keywords is the core of BDD.
-- Hint: Feature description including stories and scenarios would be the correct term.

(3) ++Systematic Analysis of Requirements++
- Vision: states what the project wants to achieve
- Business goal: describes what the business will get out of it
- Capability: expresses what users and stakeholders should be able to do to deliver these goals
- Feature: represents software functionality to support capabilities
- Story: makes clear which role wants to carry out what for which purpose
- Scenario: expresses a concrete example in a defined format by which a feature is illustrated

(4) ++Gherkin Feature and Including Scenarios++
A story consists of two parts
- Narrative description of a requirement structured according to "As a – I want – So that"
- A set of acceptance criteria presented as scenarios structured according to "Given – When – Then"

(5) ++Example: TLM Feature++
- Scenario: Successful creation of a todo list
  Given I have the possibility to create a todo list
  When I create a todo list with the title "WASA course"
  Then the todo list with the title "WASA course" is created

(6) ++Dealing With Gherkin Features++
- The developer and the stakeholder.
This leads to the requirement that features should not only be documented in the IDE of the developer but also in external format (e.g. Word or Bitbucket Wiki).



The motivation of Domain-Driven Design (DDD) is taken from Martin Fowler's foreword of Eric Evans's book "Domain-Driven Design – Tackling Complexity in the Heart of Software" [Ev04].

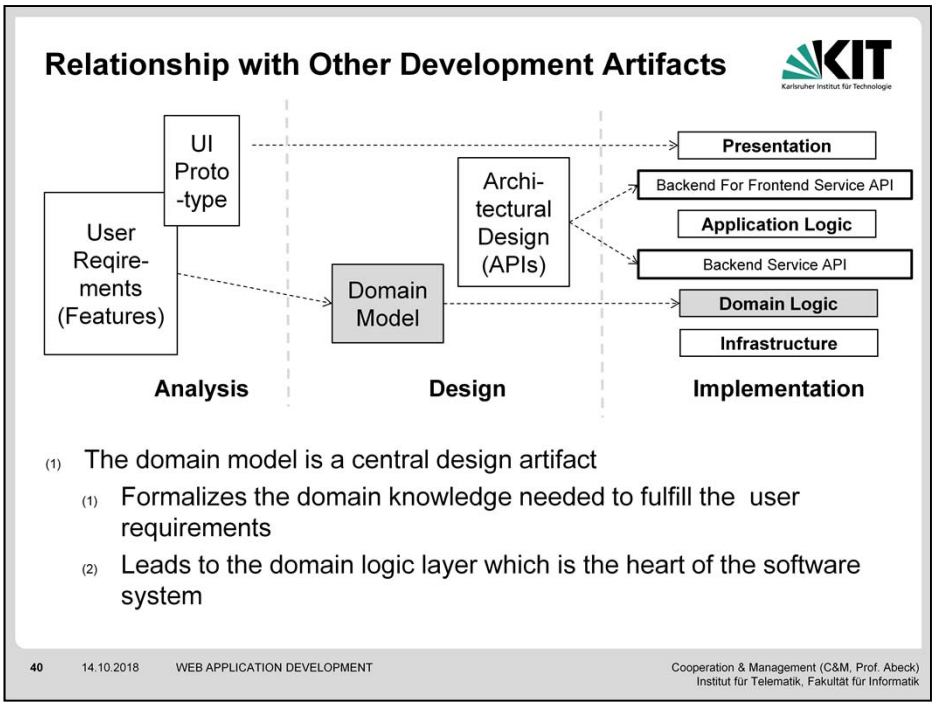
(1) A general definition of a domain is: A sphere of knowledge, influence, or activity.
(1.1) A model itself can be defined as a system of abstractions.
(1.2) By this language domain experts and technologists are tight together.

(2) There are many things that make software development complex. A deep understanding of the problem domain is in the heart of this complexity.
(2.1) An effective domain modeler can both use the whiteboard during his talk with the customer and work together with a programmer on a Java program.
(2.2) Only few people can do it well – and the competence to build good domain models is not easy to teach.
(2.3) Domain models are not first modelled and then implemented. Instead, an experienced modeler gets splendid ideas from earlier releases of the domain model.
(2.4) DDD teaches a lot of people how to apply modeling concepts to add structure and cohesion to the software development.

(3) Based on this vocabulary the activity and the hard-to-learn skill of domain modelling can be explained.

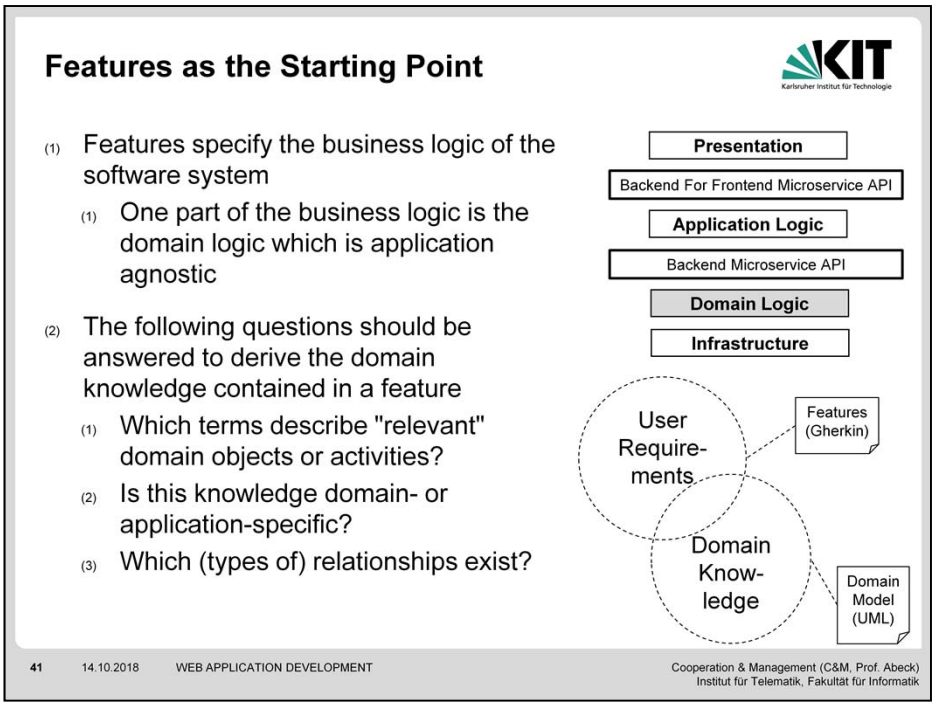
DDD Domain-Driven Design

[Ev04] Eric Evans: Domain-Driven Design – Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.



To understand how domain modeling can be practically applied it is important to understand where it can be located in the overall software development process.

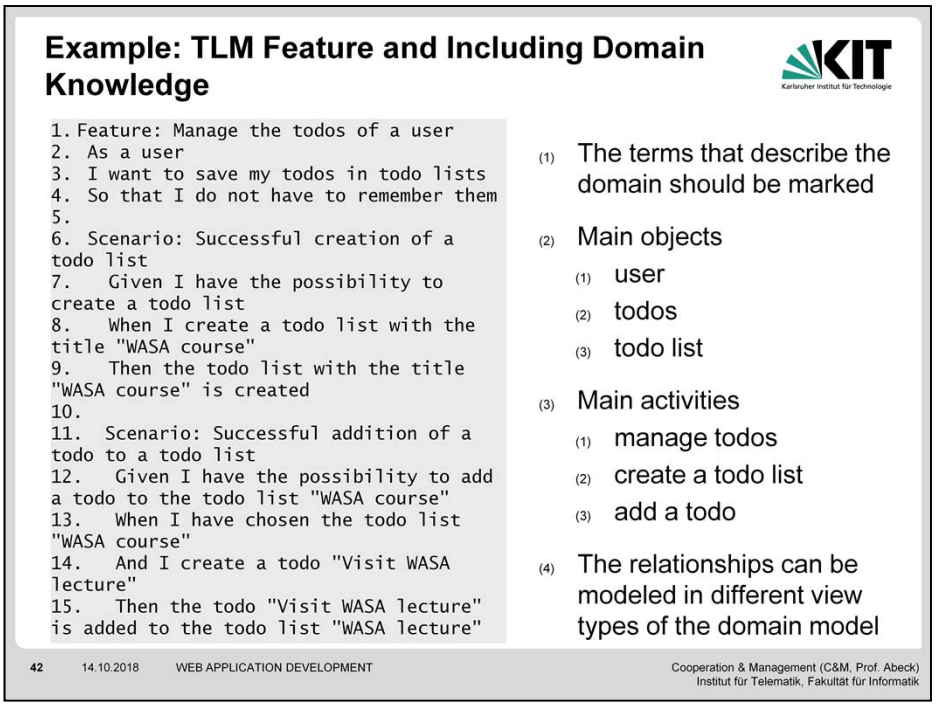
(1) The domain model is located in the design phase of a software development process. Since the domain model is used to clarify specific aspects of the domain with a domain expert (not with the user) it also contributes to the analysis phase.
(1.1) The domain model concentrates on the domain knowledge that is needed to provide the functionality of the software system. A precise and semi-formal description as a domain model is the main challenge of the design of the software system.
(1.2) Only one layer of the overall software architecture, the so-called domain logic layer is covered by the domain model. Since this layer can be seen as the heart of a software system it provides the kernel of the system. A major goal of domain modeling is to keep the model (of the domain) and implementation (of the domain logic layer) close together.



According to the overall BDD/DDD-based development process in the subsequent step the domain logic contained in features should be added to the domain model.
Note: In what follows the term feature is synonymous to a Gherkin feature.

(1) A feature specifies a specific part of the business logic from the viewpoint of a user who applies the functionality in a certain role ("As a ..." in the story) and in certain situations ("Given ..." in the scenario).
(1.1) The other part is the application logic. It is important to notice that in the DDD approach the business logic is divided into the domain logic and the application logic. The domain logic is application agnostic which means that this logic is relevant for other applications of this domain. Therefore, the domain logic is the re-usable part of the business logic appearing in a feature.

(2) Like the process of feature specification the derivation of domain knowledge from the features is a highly creative process. Nevertheless, some best practices do exist.
(2.1) By answering this question those terms of the feature should be identified that are relevant for the understanding of the user requirement.
(2.2) Only those aspects identified by Question 1 should be added to the domain model which are not specific to the regarded application.
(2.3) Three types of relationships can be distinguished: general associations, containment (composition, aggregation), inheritance

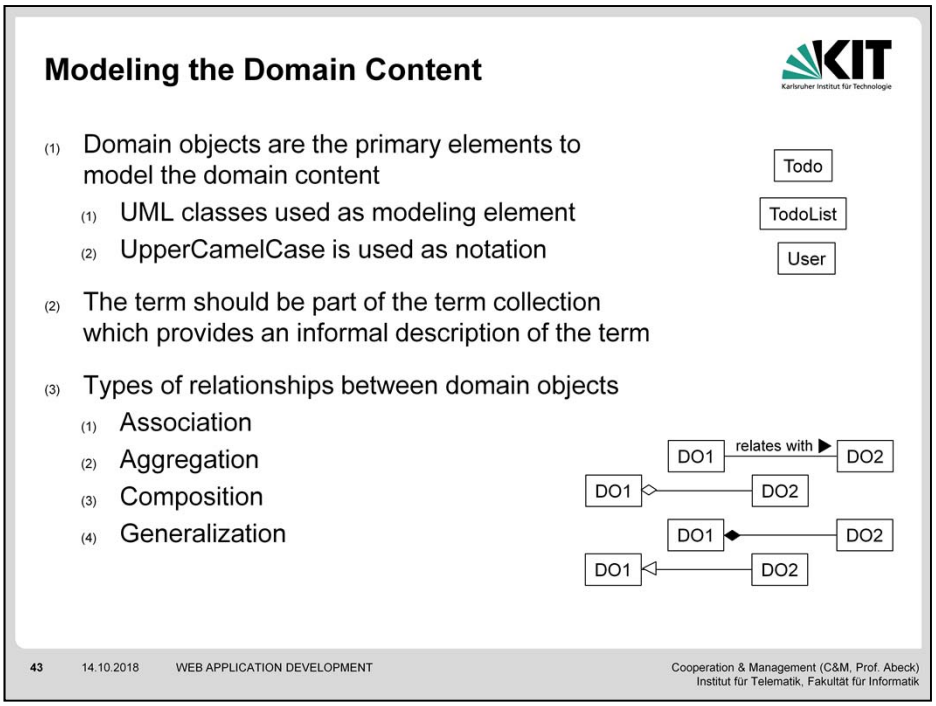


The derivation of domain knowledge is shown with a part of the TLM feature that was introduced on a previous page.

(1) The approach is to go through the lines of the feature and identify those terms which are presumably relevant for the understanding of the domain and the functionality related to the domain, i.e. the domain logic.

(2) (3) The list should be seen as a candidate of terms that might become part of the ubiquitous language of the domain. For those terms which certainly will be part of the language, a definition should be formulated.

(4) The relation view is specified as a UML class diagram as illustrated in what follows.



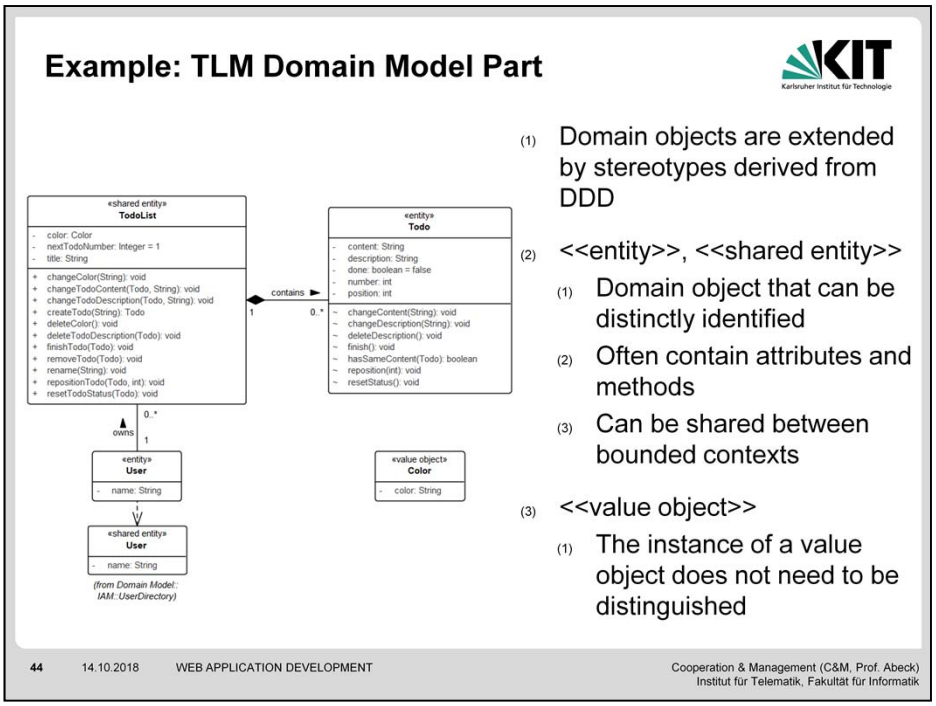
(1) In DDD, entities and value objects are the fundamental domain objects.
(1.1) The graphical notation is a rectangle which expresses the stereotype <<class>>.
(1.2) This notation is also used for all other modeling elements, such as the structuring element of a package that has been introduced before.
(Hazard Warning, PointOfInterest, Thesis) These are examples of domain objects from different domain models and software systems.

(2) The term collection is an artifact that is created in the analysis phase of the C&M software development process.

(3) The following types of relationships are standardized by UML which can be used in a class diagram.
(3.1) The association is the most general relationship between two domain objects DO1 and DO2. The association can be named (e.g. defines, is responsible for, creates). An arrow indicates the direction of the association.
(3.2) The aggregation is a stronger version of association and it is used to indicate that a domain object DO1 contains another domain object DO2 [MH06:86]. The lifetimes of the containing domain object (DO1) and the contained domain object (DO2) do not depend on each other. An aggregation is modeled by using an empty diamond arrowhead.
(3.3) The composition is as the aggregation a containment relationship, i.e. domain object DO1 contains another domain object DO2 [MH06:86]. In contrast to an aggregation, the lifetime of the containing domain object (DO1) depends on the lifetimes of the contained domain object (DO2). This means that the contained domain object (DO2) cannot exist without the containing domain object (DO1). A composition is modeled by using a filled diamond arrowhead.
(3.4) Generalization which is otherwise known as inheritance is used to describe that a domain object DO1 is a generalization of a domain object DO2. Therefore, the sub domain object DO2 inherits all properties from the super domain object DO1. The generalization relationship can be denoted as "is a" or "is a type of". It is modeled by an empty generalization arrow.

DO Domain Object

[MH06] Russ Miles, Kim Hamilton: Learning UML2.0, O'Reilly, 2006.



So far, the domain objects have been identified. In a next step, domain modeling concepts are applied in order to build the domain model. On this page, this step is shown with the example of a part of the TodoListManagement (TLM) domain model

Note: This part belongs to the so-called "bounded context" TodoManagement which is introduced in one of the next pages.

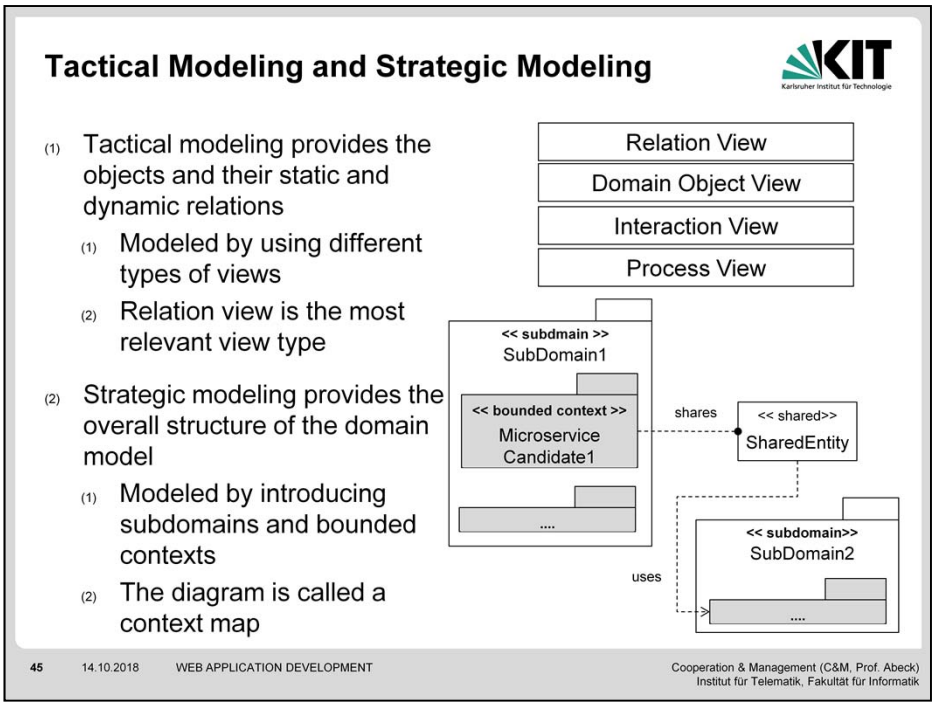
(1) The stereotypes (e.g. <<shared entity>>, <<entity>>, and <<value object>>) are defined by the Domain-Driven Design (DDD) concepts. Each domain object belongs to one or more such stereotype.

(2) Entities are domain objects that can be clearly identified. They can contain attributes and methods.
(TodoList, Todo) These are the two main domain objects of the example.
(2.1) If two Todo domain objects have the same status and contain the same information, they are the same Todo domain object.
(2.2) Example of an attribute and a method of the Todo entity are content and changeContent().
(2.3) If objects are shared between bounded contexts, the stereotype is annotated with "shared".
(<<shared entity>> TodoList) Since a TodoList entity is shared between the bounded contexts "TodoManagement" and "UserDirectory" it is modeled by using the stereotype <<shared entity>>.
(<<shared entity User) The entity User is not part of the domainTodoListManagement but it is provided by the domain IAM (Identity and Access Management).

(3) Value objects represent a descriptive aspect of the domain for which is only important what they are, but not who or which they are.
(3.1) This lack of constraint gives design freedom and simplifies to optimize performance (e.g. by copying and not sharing a value object).
(<<value object>> Color) Since it is not important which "red" was selected this information of the domain model is modeled as a value object.

Remark: Further stereotypes like aggregates or domain functions do exist but they are not explained here.

IAM Identity and Access Management



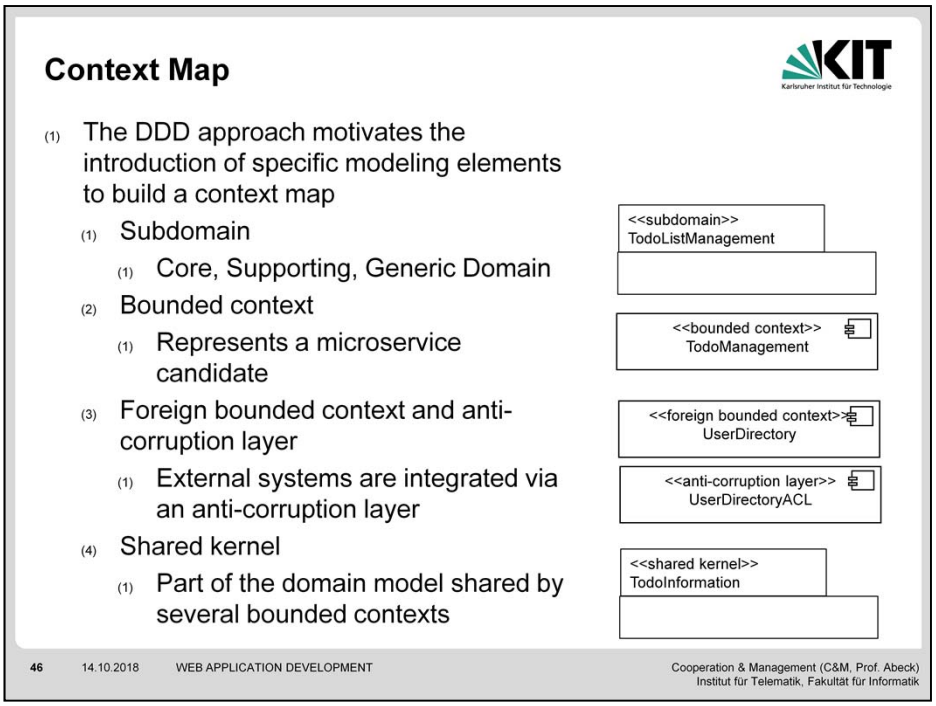
The domain model emerges from two types of modeling, the tactical modeling and the strategic modeling.

(1) The tactical modeling corresponds to the well-known proceeding of modeling knowledge as conceptual class diagrams added by dynamic diagrams to model certain time-related aspects. The result from the tactical modeling can be seen as the core content of the domain model.
(1.1) Four types of views are distinguished. Only the process view allows to model dynamic aspects, whereas the relation view, domain object view, and interaction view focus on the static aspects.
(1.2) The relation view corresponds to conceptual class diagrams as will be shown with an example.

(2) The strategic modeling is the real new part of domain modeling and the result, a so-called context map, provides the link to the microservice architecture.
(2.1) The structuring of the domain knowledge (as it is described by tactical modeling based on the four view types) is carried out in two steps: The whole domain is structured into subdomains and the subdomains are structured into so-called bounded contexts. To find an adequate structure coupling that domain knowledge which is coherently connected "from its nature" is one of the main challenges in domain modeling. The term "bounded context" is derived from the approach of a Domain-Driven Design (DDD, [Ev04]). A bounded context form the architectural perspective can be seen as a microservice candidate. This is the reason why a domain model serves as a blueprint for the microservice architecture which implements the model in the domain logic layer of the layered architecture. Domain objects can be shared by bounded contexts. These objects are called shared entities.
(2.2) DDD does not prescribe a concrete representation of the model. At C&M, the modeling elements from the Unified Modeling Language (UML, (CM-W-UNI]) are used to model the results from the strategic (and also from the tactical) modeling.
(<<package>> SubDomain1) The subdomain is modeled as a standard UML package. "SubDomain1" stands for the name of the name of the subdomain formulated in UpperCamelCase.
(<<bounded context MicroserviceCandidate1>> MicroserviceCandidate1) To be able to model a bounded context in UML a new stereotype <<bounded context>> is introduced. The UML modeling symbol of a package is used as graphical representation. "MicroserviceCandidate1" stands for the name of the bounded context formulated in UpperCamelCase.
(<<shared>> SharedEntity, uses) This is the way how domain models shared between bounded contexts. In the example, the bounded context MicroserviceCandidate1 shares the domain object "SharedEntity" with a bounded context being part of SubDomain2.

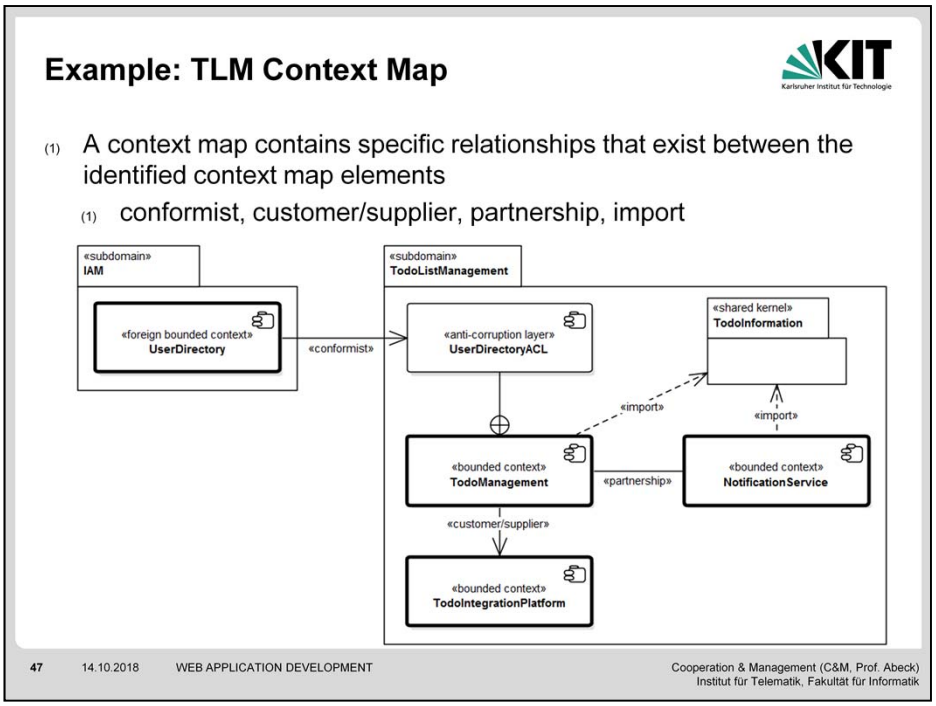
DDD	Domain-Driven Design
UML	Unified Modeling Language

[CM-W-DOM] Cooperation & Management: DOMAIN MODELING, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*
[CM-W-UNI] Cooperation & Management: UNIFIED MODELING LANGUAGE, WASA-Kurseinheit. https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA
[Ev04] Eric Evans: Domain-Driven Design – Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.



The context map is a diagram which can be seen as the starting point of domain modeling since it provides the domain structure in which the tactical domain models are placed in a systematic and consistent way. A context map does not focus the domain content (i.e. domain objects and its relationships), but a coherent structure of the business functionality.

(1) The modeling elements that can be used in a context map are introduced in what follows.
(1.1) By introducing subdomains the overview of a larger domain can be kept. Each subdomain should have a size which can be easily overlooked.
(<<subdomain>> TodoListManagement) This is the core subdomain of the TodoListManagement domain. A stereotype <<subdomain>> is introduced which uses the UML modeling element of a package as graphical representation.
(1.1.1) Core subdomains are in the center of the context map since they are the building blocks of the domain which is to be described. The TodoManagement subdomain is part of the core. Supporting and generic subdomains provide some additional functionality that are needed. An example of such a subdomain in the TodoListManagement domain is the IAM subdomain.
(1.2) (1.2.1) A bounded context packages coherent functionality in a component. In the software architecture derived from the context map a bounded context is a candidate for a microservice.
(<<bounded context>> TodoManagement) This bounded context is part of the subdomain TodoListManagement. The stereotype <<bounded context>> is represented as a UML PackagingComponent. This allows to use UML compliant associations to model different types of relationships between bounded contexts as described on the next page.
(1.3) The modeling elements of a foreign bounded contexts and an anti-corruption layer allow to model functionality from foreign domains.
(<<foreign bounded context>> UserDirectory) (<<anti-corruption layer>> UserDirectoryACL) The user directory is part of the IAM (sub) domain. Therefore, this functionality is modeled as a foreign bounded context which is integrated into the core functionality by using an anti-corruption layer called UserDirectoryACL.
(1.4) If there is an overlap of a part of the domain model in two or more bounded contexts it might make sense to model this overlap in a shared kernel.
(<<shared kernel>> TodoInformation) Since a shared kernel can be seen as a kind of directory and not as a running software component, it is modeled a a package. In the example, the shared kernel TodoInformation describes the domain logic with respect to the todos which are shared by two bounded contexts, TodoManagement and Notification.



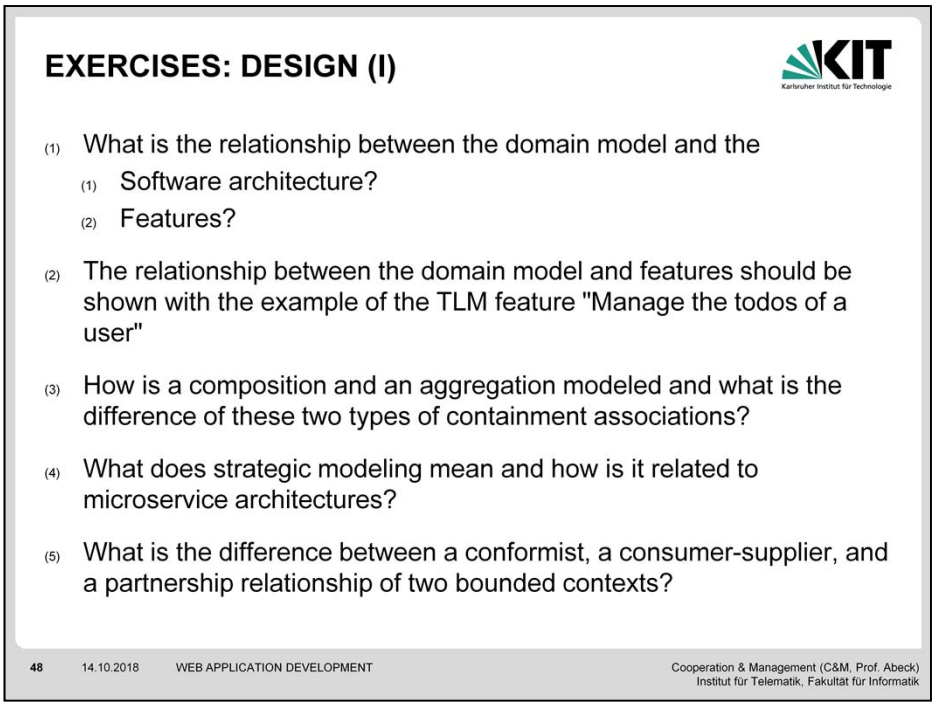
(1) Not only the modeling elements (e.g. subdomain or bounded context) but also the relationships between these modeling elements are described by a context map.
(1.1) As the modeling elements itself the relationships between them are specific to a context map. In the following each of the four specific relationships are described with the exemplary context map.

(<<conformist>>) This type of relationship is a directed association between two bounded contexts. The bounded context building the source of the directed association is called downstream (in this case, UserDirectory) and the destination (UserDirectoryACL) is called upstream. In a conformist relationship the downstream must conform to (i.e. strictly obey) any changes made by the upstream even if the model and the implementation from the upstream does not fit to the downstream.

(<<customer/supplier>>) A directed association between two bounded contexts in which the downstream is the customer (in the example, TodoManagement) and the upstream is the supplier (TodoIntegrationPlatform). The association means that the upstream team which develops the TodoIntegrationPlatform must coordinate planned changes with the downstream team which develops the TodoManagement.

(<<partnership>>) An undirected association between two bounded contexts. A partnership association means that the two teams responsible for the bounded contexts depend on each other and must, therefore, cooperate closely. In the example, this is the case for the bounded contexts TodoManagement and Notification although the shared kernel TodoInformation might lead to a looser coupling.

(<<import>>) This directed association is used between a bounded context and a shared kernel which needs this part of the domain model described by the shared kernel. In the example, the shared kernel is imported – and in this way, shared – by the two bounded contexts TodoManagement and NotificationService.



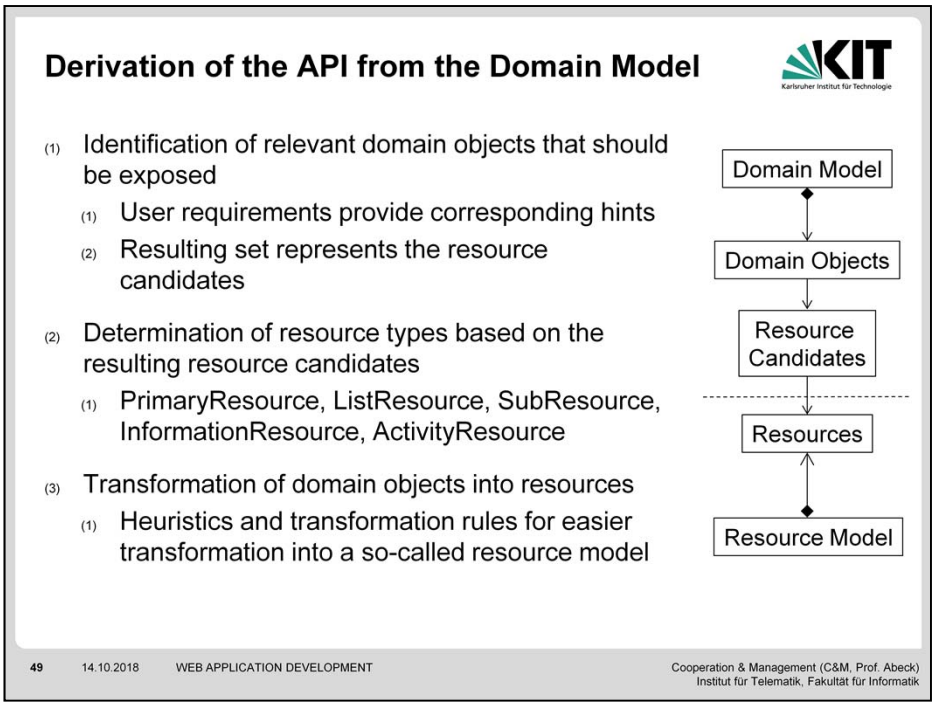
(1.1) ++Domain Modeling and Software Development++
- The domain model provides the functionality of the domain logic layer.
(1.2) ++From Features to the Domain Model++
- Features contain domain knowledge which should be formalized in a domain model according to DDD
- Features specify that part of the business logic of the software system that should be implemented in the domain logic layer and which should be separated from the application logic layer.

(2) ++Example: TLM Feature and Including Domain Knowledge++
- The approach is to go through the lines of the feature and identify those terms which are presumably relevant for the understanding of the domain and the functionality provided by the domain, i.e. the domain logic.
- In the TLM examples these are
-- Objects: user, todos, todo list
-- Activities: manage todos, create a todo list, add a todo

(3) ++Modeling the Domain Content++
- Composition: filled diamond arrowhead
- Aggregation: white diamond arrowhead
- Difference: In contrast to an aggregation, the lifetime of the containing domain object (DO1) depends on the lifetimes of the contained domain object (DO2)
- Example: TodoList is a composition of Todos -> Deletion of the TodoList leads to the deletion of all contained todos
TodoList is an aggregation of Todos -> Deletion of the TodoList does NOT delete the contained Todos

(4) ++Tactical and Strategic Modeling++
- Strategic modeling provides the overall structure of the domain model by defining subdomains and bounded contexts.
- Relationship: Bounded contexts are the candidates for microservices.

(5) ++Example: TLM Context Map++
- <<conformist>>: This type of relationship is a directed association between two bounded contexts.. In a conformist relationship the downstream must conform to (i.e. strictly obey) any changes made by the upstream even if the model and the implementation from the upstream does not fit to the downstream.
- <<customer/supplier>>) The association means that the supplier must coordinate planned changes with the custmer.
-<<partnership>>) A partnership association means that the two teams responsible for the bounded contexts depend on each other and must, therefore, cooperate closely.



Besides the domain model the Application Programming Interface (API) is an important design artifact. The API is specified in the second phase of the design phase. Both artifacts, the domain model and the API, are strongly related.

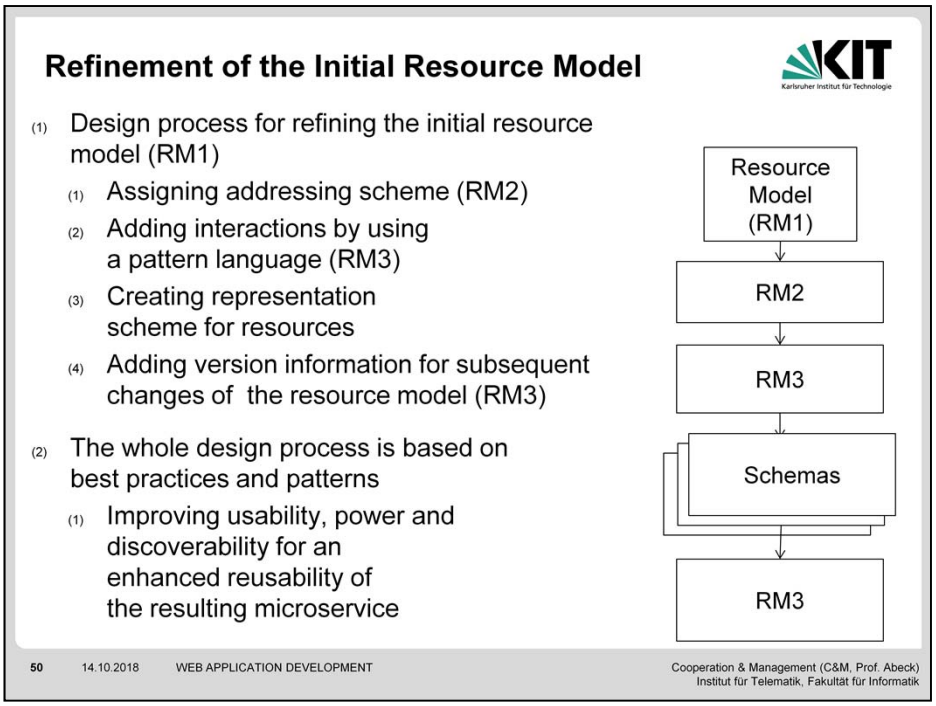
(1) As has been shown, the features as the central analysis artifact are a good basis to determine the domain objects.
(1.1) For instance, an explorative prototype, based on the user requirements, shows which interactions are needed.
(1.2) The output of this step is called "candidates". For each candidate, the software architect has to choose if the domain object is exposed or not. They are transformed into "real" resources by a transformation described in the following steps.

(2) Based on the resource candidates the resource types are determined.
(2.1) The semantics of each resource type are described in more detail in [Gi18]. The resource types used in this approach are based on the current research results in this field:
- PrimaryResource: A resource which represents a domain object and which is directly addressable. A primary resource can be derived from an object of the domain model that has no inbound associations.
- SubResource: A resource which is a logical part of an other resource and which is directly addressable. These are resources which have one or more inbound associations.
- ListResource: A resource which contains more resources as list elements.
Domain object with multiple instances or which is in a 1..n relationship and which represents the parent.
- InformationResource: A resource that can not dereference itself but is associated with an identity. A domain object which is represented as a value object and which has a inbound association.
- ActivityResource: A resource that represents a business process. Domain object which is modeled as a domain service.

(3) The semantics of each resource type as well as the domain objects and their associations allow to derive heuristics and transformation rules.
(3.1) The heuristics identify a resource type based on the domain model. But only the resource candidates have to be considered for identification. Each heuristic is linked with a transformation rule that transforms a resource candidate into a "real" resource by assigning a stereotype from a dedicated UML profile called resource orientation and placing it into a so-called resource model. For example, the heuristic for a primary resource is the following: A domain object is an entity and has no outgoing associations. The transformation rule for this heuristic consists of these steps: The first step is to copy the entity into the resource model and remove all existing stereotypes, except the identifier. The next step is adding the stereotype <<PrimaryRessource>> to the copied entity. The resource model applies the previously mentioned UML profile and extends the UML with concepts of REpresentational State Transfer (REST). It is important to note that also behavioral aspects can be modeled within the resource model. To keep the model lightweight, CRUD operations were not modeled and instead considered as implicitly given.

API	Application Programming Interface
REST	REpresentational State Transfer

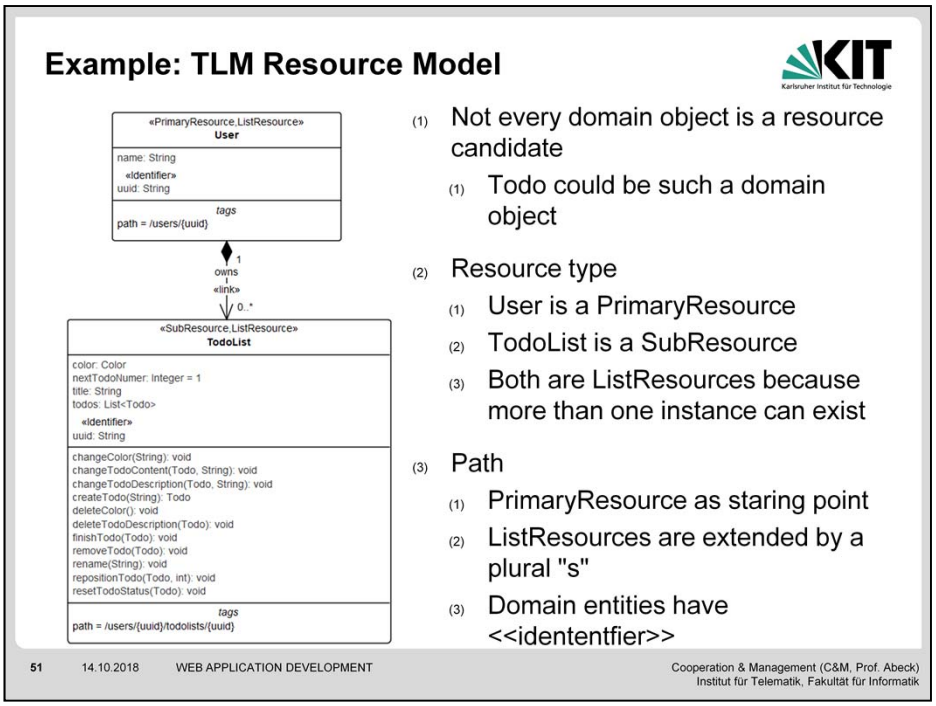
[Gi18] Pascal Giessler: Domänengetriebener Entwurf von ressourcenorientierten Microservices, Dissertation, Karlsruhe Institute of Technology (KIT), Cooperation & Management (C&M, Prof. Abeck), 2018. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-5.Publikationen/1.Vergangene_Jahre/1.Promotionen*



(1) The initial resource model (called RM1 in the refinement process) is created based on the underlying domain model, as showed on ++Example: NavSG Resource Model++. In addition to the structural aspects, the behavior-specific aspects of the domain are also transferred in a suitable form so that the behavior is also reflected in the resource model. For a complete and detailed description of these steps see [Gi18].
(1.1) Each resource is supplemented by a Uniform Resource Locator (URL) component based on its assigned resource type. By following links between resources, the complete URL can be derived. For the creation of corresponding components, linguistic patterns and best practices are used to increase the usability and ultimately the reusability. An example is showed on slide ++Example: NavSG Resource Model++.
(1.2) After adding addressing information to RM2, the interactions with a resource from the client is considered. The offered behavior by the domain has to be mapped onto a uniform interface which is a constraint of the architectural style REST. To simplify the mapping, existing patterns were identified and assigned to the individual resource types. Regarding the resulting influence, each pattern was extended by their influences on quality characteristics. Since the web API is the link between a client and a resulting microservice, the patterns also show their effect on different interaction participants. After this steps, RM3 has been created.
(1.3) The creation of representation schemas is the next logical step. The schemas define the structure of the request/response body. In addition to listing the required and optional attributes, they can also be used to define valid attribute values or to show sample values of some attributes. Based on these schemas, mock values can be generated that may be suitable for web API testing.
(1.4) The last step represents the versioning of the resource model and finally the web API. Version information using Semantic Versioning (SemVer) is added to RM3. Since this is only a small addition from the technical perspective it is still an RM3. The representation of this version information in the web API is treated by patterns. Besides, a process is introduced on how to change the version information. According to the versioning process more than one resource model may exist at the same time. RM3 is the final result and an example is showed on ++Example: NavSG Web APIs (Excerpt)++.

(2) Each refinement step can be found in many of today's web APIs. The refinement process combines the expert knowledge and forms a systematical and traceable process to build well-designed web APIs for the modeled domain. Web APIs are an important asset of digital transformation and have to be designed with care.

API	Application Programming Interface
RM	Resource Model
REST	REpresentational State Transfer
URL	Uniform Resource Locator



The TLM resource model is derived from the TLM domain model which was described on a previous page ++Example: TLM Domain Model++.

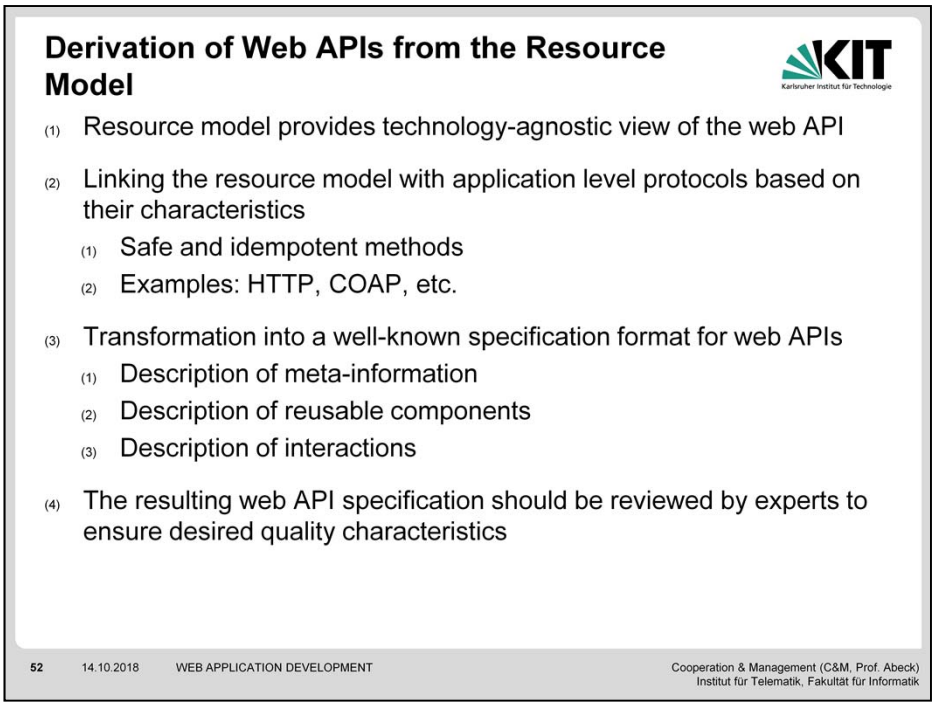
(1) Not every domain object is a resource candidate. That is because not every domain object can or should have public access. There is a variety of reaons for that.
(1.1) In the TLM application Todo might be completely accessed via methods from TodoList. In this case the domain object Todo must to be made public and it would not become a resource.
Remark: In the API introduced in one of the next pages Todo is handled as a resource. This means that a different design decision has been taken.

(2) The resource candidates are assigned stereotypes. Those types can be PrimaryResource, SubResource, ListResource, InformationResource or ActivityResource.
(2.1) In TLM User is a PrimaryResource since it is a domain object without inbound relations.
(2.2) Every object which is associated with a PrimaryResource is a SubResource. Further objects with inbound relationships are also SubResources. TodoList is a SubResource of User.
(2.3) A resource which can have more than one instance is categorized as a ListResource. That is the case for both resources, User and TodoList, in the example.

(3) The path by which the resource can be accessed is built using an algorithm described in [Gi18, Hü18].
(3.1) The starting point of the path is a PrimaryResource. Afterwards all SubResources of that resource are handled.
(3.2) If it is a List Resource a pural "s" is extended. Otherwise the singular is concatenated.
(3.3) If the resource was an entity in the Domain Model, it can be identified by an ID. The ID is appended to the path as well. It is denoted with curly braces which means that its value is defined at runtime.

(<<PrimaryResource ...) The class diagram shows the resource model of TLM. The resources have a stereotype (like PrimaryResource), attributes, methods and a path tag which denotes the path under which the resource can be accessed.

[Hü18] Tobias Hülsken: Eine Microservice-Architektur für das Identitäts- und Zugriffsmanagement, Masterarbeit, Karlsruher Institut für Technologie (KIT), C&M (Prof. Abeck), 2018.



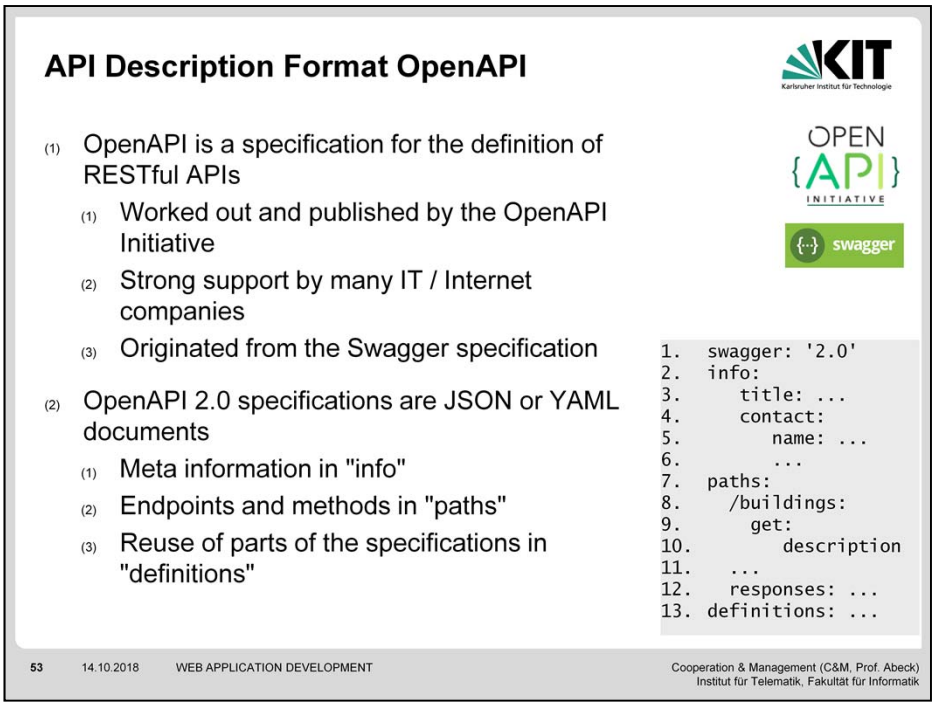
(1) The resource model itself is the starting point for deriving a web API. The resource model does not contain any design decisions regarding the technology or platform. Therefore the developer can choose any technology during the further development. The technology decisions can be made independently by the respective development teams.

(2) The choice of the appropriate protocol is based on their characteristics which have to comply with the needs of the resource model. In doing so the application level protocol should fit to the service consumers and if necessary their non-functional requirements. Some issues for this choice are the connection type (synchronous or asynchronous), the required exchange format and required methods.
(2.1) Safe methods are methods that do not modify resources. Idempotent methods are methods which outcome won't change no matter how often they are called. HTTP safe methods are GET, OPTIONS and HEAD, while idempotent methods are GET, PUT, DELETE, OPTIONS and HEAD.
(2.2) Examples of protocols for accessing resources are HTTP and COAP (Constrained Application Protocol). While COAP is mainly used for machine to machine communication in unreliable environment, HTTP is the most used application layer protocol.

(3) To make further use of the web API it can be transferred into a machine interpretable specification format. By this independent testing of the backend and frontend, automated building of the API and an automated code template creation can be enabled. Examples for machine interpretable API specifications are Open API (formerly Swagger) or RAML. For practical use at least the following has to be done.
(3.1) The meta information includes its version, general information (title and description) as well as contact information for support queries.
(3.2) Furthermore, reusable components have to be described. These components could be schemas, responses and request parameter. For instance, a query parameter description consists of a name, a short explanation, and a data type.
(3.3) An interaction can be composed of the reusable components.

(4) If the API is used in a large environment, it is very important that it is as stable as possible and meets a certain quality standard. Otherwise the API becomes less useful and generates more costs as actual benefit. Therefore an review can be performed to check its quality criteria. Possible reviewers are the API consumers as well as dedicated experts in the API design field. If there are required actions, it is necessary to populate it in the resource model and propagate it through the entire process again.

COAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
RAML	RESTful API Modeling Language



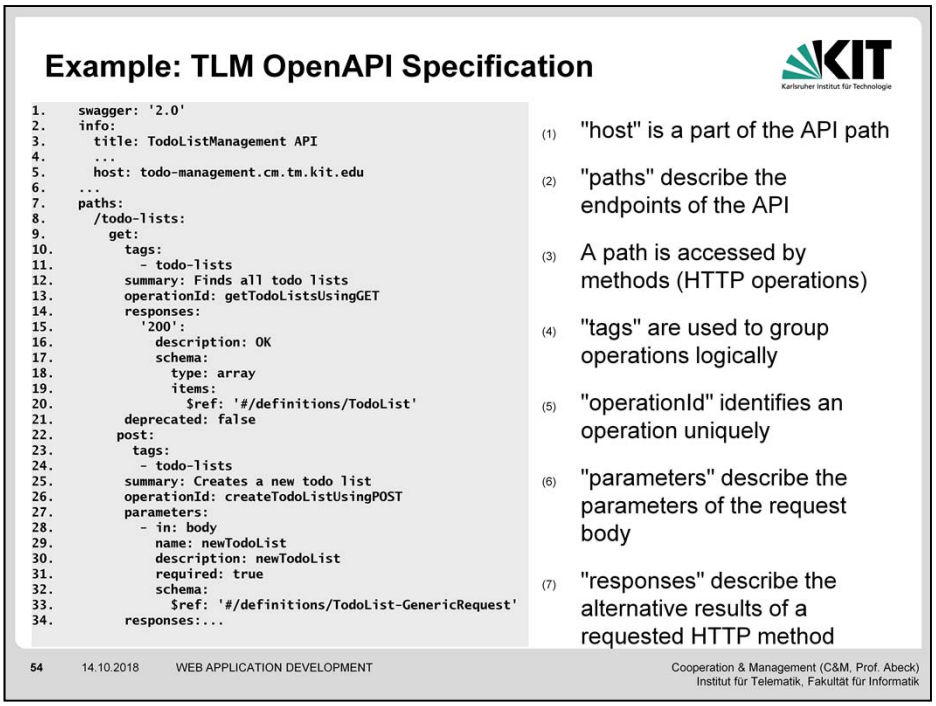
OpenAPI is one of the most broadly accepted API description formats.

(1) The OpenAPI Specification is a manufacturer-independent defacto standard.
Examples of other description formats are API Blueprint or RESTful API Modeling Language (RAML).
(1.1) The OpenAPI Initiative was founded in 2015 as an open source project under the Linux Foundation [OAI-FAQ]. A major driver was SmartBear which donated the Swagger Specification to the OpenAPI Initiative
(1.2) Examples of such companies are Google, Microsoft, IBM, Apigee.

(2) JSON and YAML are both markup language which are based on the same concept (lists and scalars). Each JSON document is a valid YAML document. The first line of the document (i.e. swagger: '2.0') defines that this is an OpenAPI 2.0 specification.
(2.1) In the "info" JSON/YAML object, metadata of the OpenAPI specification can be found, such as the "title" or the "contact" person of the service. All information is described by a cascaded list.
(2.2) In "paths" the (relative) URLs of the endpoints and the HTTP methods are indicated. In "responses" the results including the HTTP status codes are described.
(2.3) Reusable parts are objects that consist of a set of properties. To be able to reuse the object a name is defined which is use as a reference.

JSON	JavaScript Object Notation
RAML	RESTful API Modeling Language
YAML	YAML Ain't Markup Language

[OAI-FAQ] Open API Initiative: FAQ. *https://www.openapis.org/faq*



The YAML document of the TLM OpenAPI specification is available on the Bitbucket repository [CM-B-swa].

(2. info) The "info" part contains general information (such as title, description, contact) that describe the TLM API.

(1) (5. host ...) This is a central part of the API path [SWA-API] which is built as follows:
<operation> (e.g. GET) <scheme> (HTTP) <host> (todo-management.cm.tm.kit.edu) <path> (todo-lists)

(2) (7. paths:) An endpoint corresponds to a resource that the API exposes. An examples of endpoint or resource of the TLM API is "/todo-lists" (line 8.).

(3) (9. get:) (22. post:) As can be seen in the example, a single path can support multiple operations. In this case, GET /todo-lists and POST /todo-lists.

(4) (10. tags:) (23. tags:) For example, SwaggerUI (see also page ++Example: TLM API++) uses "tags" to group the displayed operations. Often the endpoint names are used as tags which is also done in this example.

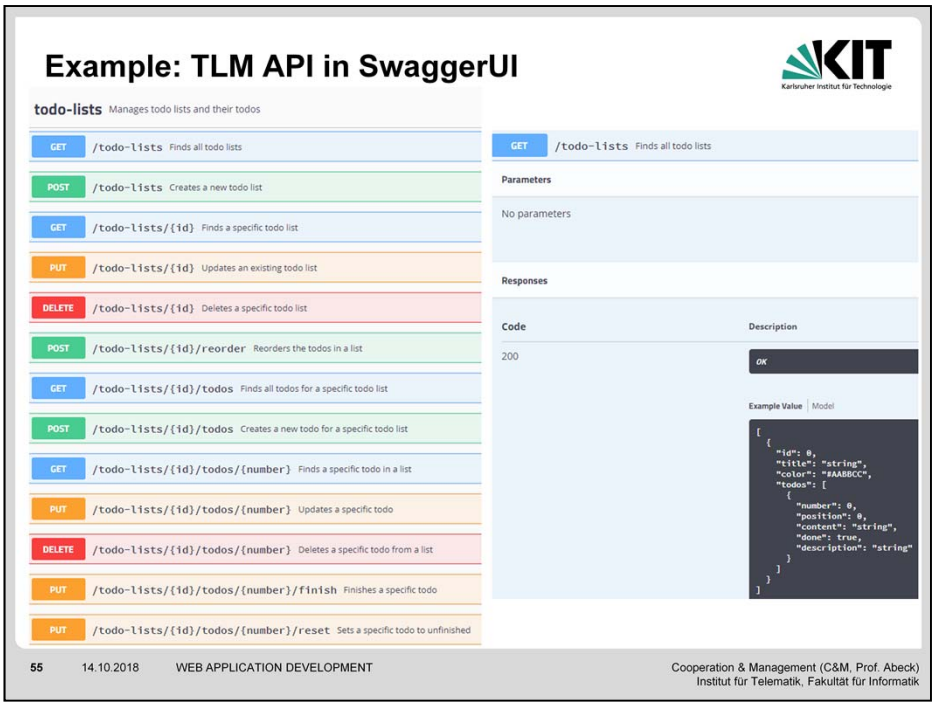
(5) (13. operationId: getTodoListUsingGet) (26. operationId: createtodoListUsingPOST) These are the names of the corresponding methods in the backend code. Usually, the (optional) operationIds are automatically generated, i.e. when the OpenAPI specification is generated from the code).

(6) (27 parameters:) If an operation (such as POST or PUT) has parameters in its request these are described as body parameters ("- in: body", line 28). In the example, the parameter "newTodoList" is specified as body parameter of the operation POST /todo-lists by which a new todo list is created.
(32. schema:) (33. $ref: '#/definitions/TodoList-GenericRequest') The schema describes the ... The "$ref" indicates that for modularity purposes this description is part of the "definitions" section of the OpenAPI specification.

(7) (14. responses) (34. responses) In this part of the "paths" object the possible results of the get operation are described. In the example a successful (200, line 15) is defined.

($ref ...) This is a reference to the "definitions" part of the OpenAPI specification where the different data structure related to the domain objects (e.g. Todo or TodoList) are defined.

[CB-B-swa] Cooperation & Management: swagger.yml. *https://bitbucket.org/cm-tm/0_todolistmanagement/src/master/swagger.yml*
[Swa-API] Swagger: API Host and Base URL. *https://swagger.io/docs/specification/2-0/api-host-and-base-path/*



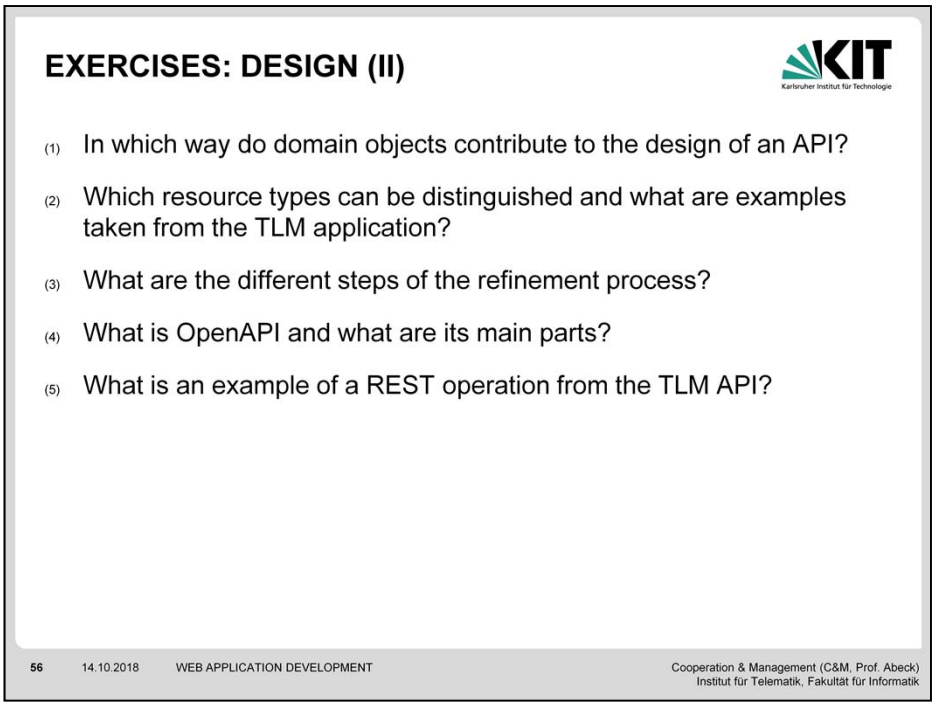
For the specification of the REST-based TLM API the tool Swagger was used [CM-S-TLM]. The page shows screen dumps taken from the tool SwaggerUI.

(GET /todo-lists) This HTTP GET operation of the TLM API returns all todo lists and all todos for each todo list.
The complete request URL is: "https://todo-management.cm.tm.kit.edu/todo-lists"

When clicking on a list element representing an operation, details of this operation is presented. The screen dump on the right hand side shows the details of the TLM API operation "GET /todo-lists".
(Parameters) Since is TLM API operation is based on a HTTP GET it has no input parameters.
(Responses) The response of a successful request (i.e. "Code 200" meaning "OK") is listed in the text field underneath "Example Value".
("id", "title", "color", "todos") These are the elements which define a todo list.
("number", "position", "content", "done", "description") A todo is defined by these five elements.

(POST /todo-lists/ ...) (GET /todo-lists/{id} ...) In what follows all REST-based operations of the TLM API are listed. For each operation a short text describes what is performed by the operation. For example, by sending a request "POST/todo-lists" a new todo list is created and a request "GET /todo-lists/{id}" finds the todo list with the specific "id" (or provides an error response if a todo list with the "id" does not exist).

[CM-S-TLM] Cooperation & Management: TLM API, Swagger Specification. *https://todo-management.cm.tm.kit.edu/swagger-ui.html*



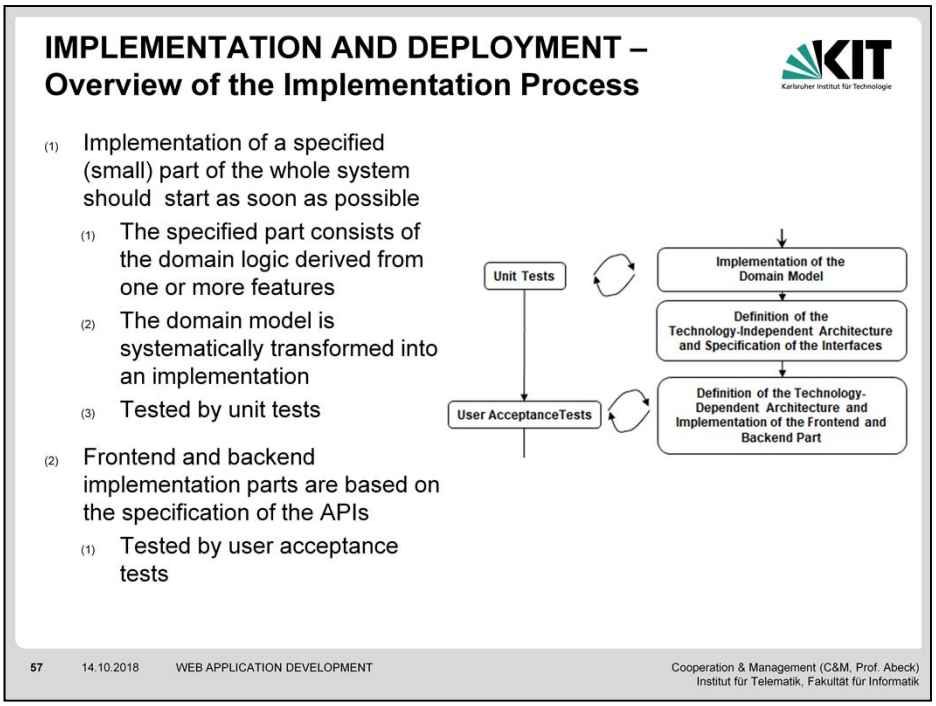
(1) ++Derivation of the API from the Domain Model++
- Domain objects are candidates for the resources which are the main building blocks of an API (= resource-oriented web API)
- Those domain objects which should be accessed via the service interface become API resources

(2) ++Derivation of the API from the Domain Model++ ++Example: TLM Resource Model++
- PrimaryResource: A resource which represents a domain object and which is directly addressable. A primary resource can be derived from an object of the domain model that has no inbound associations.
-- TLM example: User
- SubResource: A resource which is a logical part of an other resource and which is directly addressable. These are resources which have one or more inbound associations.
-- TLM example: TodoList
- ListResource: A resource which contains more resources as list elements.
Domain object with multiple instances or which is in a 1..n relationship and which represents the parent.
-- TLM example: User, ListResource
- InformationResource: A resource that can not dereference itself but is associated with an identity. A domain object which is represented as a value object and which has a inbound association.
- ActivityResource: A resource that represents a business process. Domain object which is modeled as a domain service.

(3) ++Refinement of the Initial Resource Model++
- Assigning addressing schema heuristics and transformation rules for addressing
- Adding interactions by using a pattern language
- Creating representation schemas for resources
- Adding version information for subsequent changes

(4) ++API Description Format OpenAPI++
- OpenAPI is an API description format published by the OpenAPI Initiative
- JSON or YAML-documents
- Main parts
-- info: metadata such as the "title" or the "contact" person of the service. All information is described by a cascaded list.
-- paths: the (relative) URLs of the endpoints and the HTTP methods are indicated. In "responses" the results including the HTTP status codes are described.
-- definitions: reusable parts are objects that consist of a set of properties. To be able to reuse the object a name is defined which is use as a reference.

(5) ++Example: TLM API++
- GET /todo-lists: returns all todo lists and all todos for each todo list
- POST/todo-lists: a new todo list is created
- GET /todo-lists/{id}: finds the todo list with the specific "id" (or provides an error response if a todo list with the "id" does not exist).

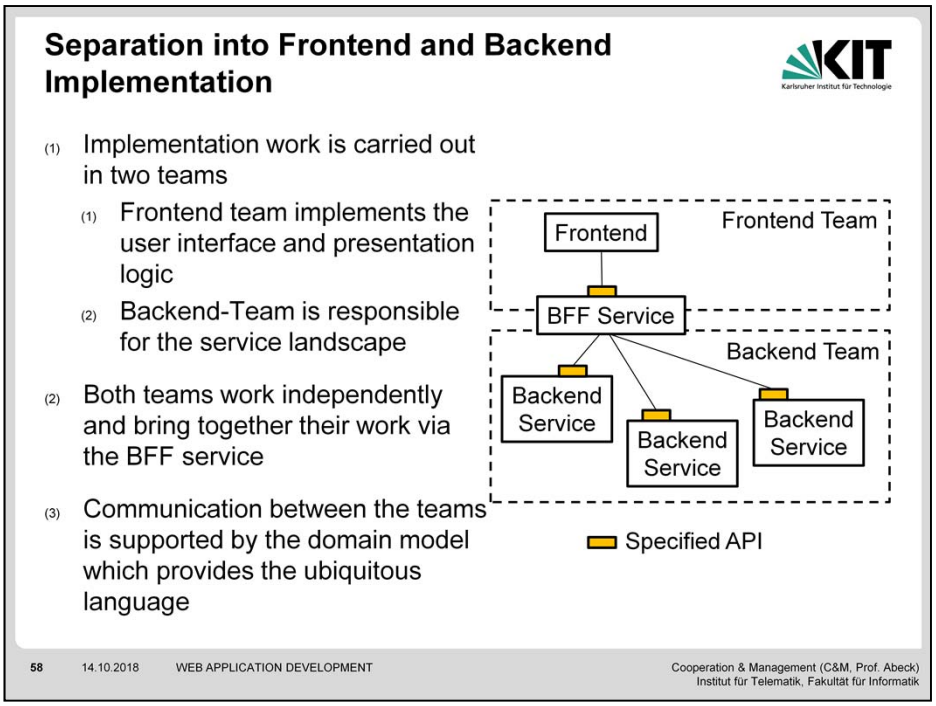


In the subsequent implementation phase the functionality as is was analyzed and designed is coded based on a microservice architecture and frameworks supporting the frontend and backend implementation. All implementation work is carried out in a test-driven manner based on unit tests and user acceptance tests which are defined by the scenarios being part of the Gherkin features. In the deployment phase the tested application is deployed in a Docker-based execution environment. As the microservice approach is strictly followed, the concept of Continuous Integration/Deployment/Delivery (CI/CD) can be supported.

(1) The BDD/DDD-based development process (Behavior-Driven Development / Domain-Driven Design) is an agile approach which means that implementation of the software should start as soon as possible.
(1.1) A (Gherkin) feature specified according to the BDD approach usually is the part of the software which is incrementally designed, implemented and tested.
(1.2) This part of the implementation concentrates on the domain logic being part of the domain logic layer of a microservice architecture. It can be carried before the APIs of the architecture are designed.
(1.3) Unit tests check if the implementation of the domain model fulfills all domain-specific assertions included in the domain model.

(2) This is the main part of the implementation work. A prerequisite for the start of this implementation is the existence of the API specifications of the microservice architecture.
(2.1) User acceptance tests are carried out by testing the scenarios occurring in the feature. In order to be able to test a feature the step definitions occurring in the features must be implemented.

BDD	Behavior-Driven Development
CD	Continuous Delivery, Continuous Deployment
CI	Continuous Integration
DDD	Domain-Driven Design

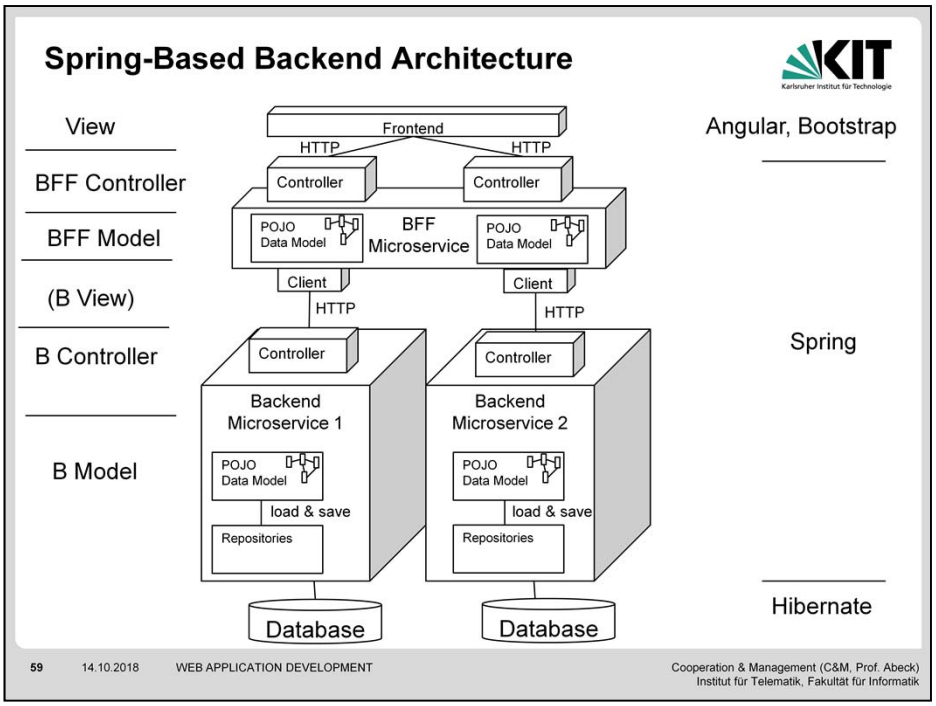


The implementation of a web application can be separated into two parts, the frontend implementation (client side) and the backend implementation (server side).

(1) Since the tasks that have to be carried out during frontend and backend application require different competences and the use of different tools it makes sense to build two separate teams.

(2) The backend for frontend (BFF) service bridges the gap between the requirements of the frontend to the backend and the available backend services.

(3) The domain model and the ubiquitous language define the common language of both teams. If changes of the domain model become necessary during the development a clear agreement between the two teams is necessary.



The figure illustrates the overall implementation architecture which is based on the MVC pattern (Model-View-Controler). The focus of the illustrated architecture is on the communication of a Backend-For-Frontend (BFF) microservice with Backend (B) microservices (Backend Microservice 1 and Backend Microservice 2) using the Spring technology.
Note: Each of these microservices (BFF and B) run on different web servers which communicate via HTTP.

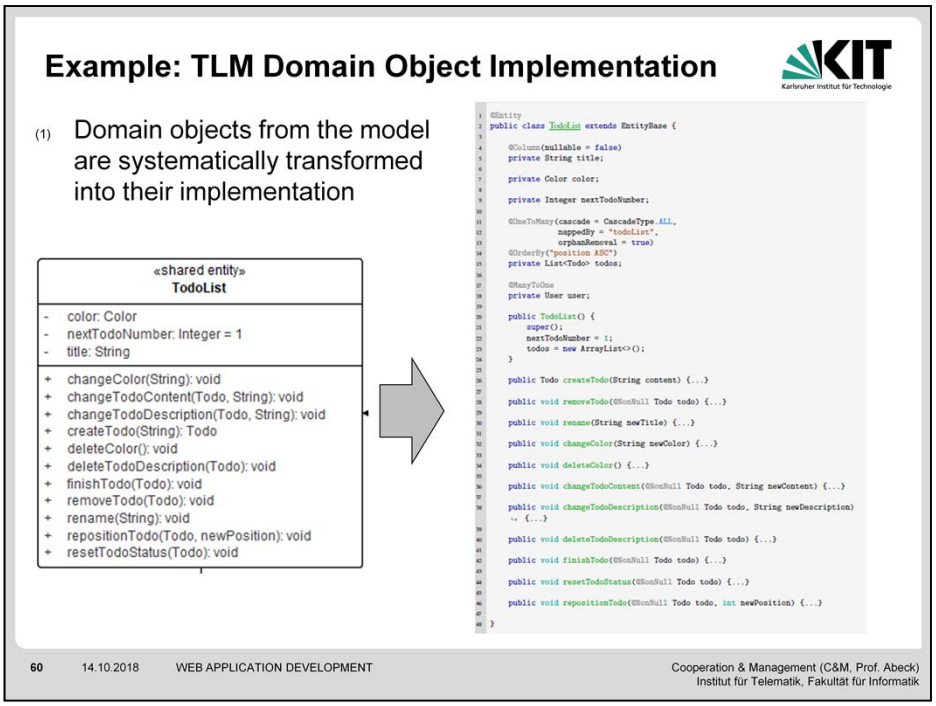
(View) The view in a microservice architecture is realized by the frontend. Technologies that can be used for the frontend implementation are Angular and Bootstrap.

There are two types of controllers in the architecture:
(BFF Controller) This is the controller directed to the frontend. It supports the frontend by transforming data for the view.
(B Controller) This is the controller related to a microservice. It starts the request and orchestrates the services.

There are also two types of models in the architecture:
(BFF Model) This is the data model that is underlying the BFF microservice.
(B Model) Each B microservice has its own data model and its own database to persist the data specific to the one microservice.

B	Backend
BFF	Backend For Frontend
MVC	Model View Controller
POJO	Plain Old Java Object

[CM-TL-Spr] SPRING, Technischer Leitfaden. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-2.Leitfäden/Technologien*



(1) The example taken from [Ir18] shows how the TodoList which is a central TLM domain object is systematically transferred into its implementation. Several JavaX annotations (e.g. @Entity, @Column, @OneToMany [Ora-JEE]) are used to solve the persistency issues of the implementation.

(<<shared entity>> TodoList) The TodoList is a <<shared entity>> which means that it is owned by one bounded context and used by at least one other bounded context. This domain object is defined by three (alphabetically ordered) attributes, color, nextTodoNumber, and title, and a number of methods (e.g. createTodo).

(1 @Entity 2 public class TodoList ...) This is the source code of the implemented domain object TodoList. Only the packet imports and the method bodies are left out.

(1 @Entity) This JavaXa nnotation is used for persistency purposes. The class is marked as an entity when it is written into the database.

(4 @Column ...) Specifies the mapped column for this property.

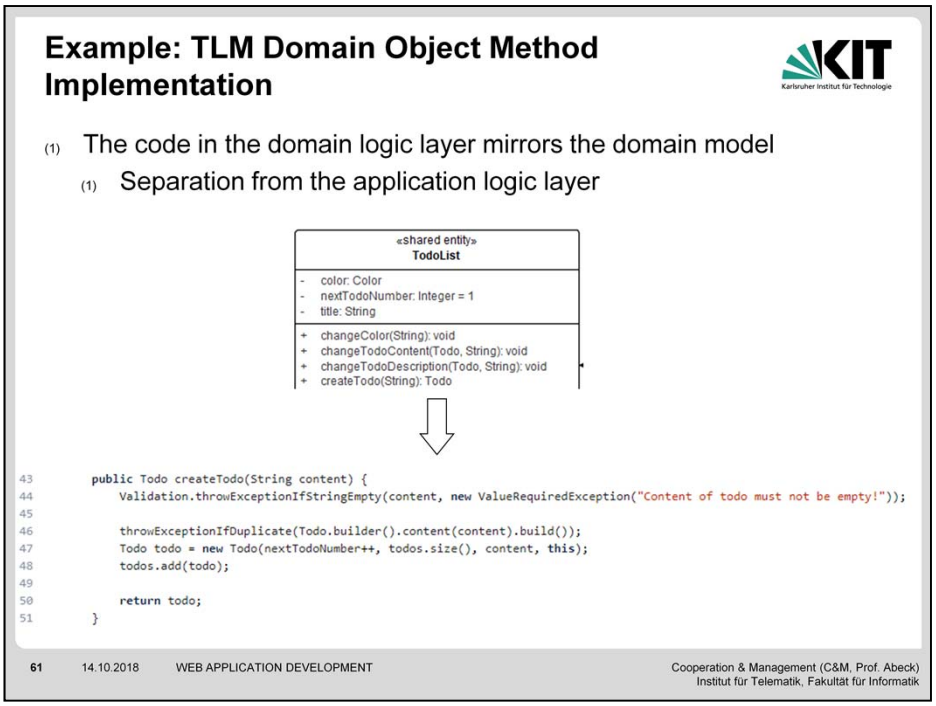
(11 @OneToMany ...) The class TodoList can be mapped to many Todos. The annotation @ManyToOne works the other way round.

(14 @OrderBy ...) The mapped objects are ordered by ascending positions.

(20 public TodoList () ...) The constructor has no parameters. New created TodoLists are empty.

(26 createTodo(String content) ...) This is the first method provided by the class TodoList. In contrast to the model of the domain object the methods are not alphabetically, but logically ordered.

[Ir18] Chris Irrgang: Eine Microservice-Architektur zur Bereitstellung innovativer Mobilitätsdienste in der Connected-Car-Domäne, Masterarbeit, Karlsruher Institut für Technologie (KIT), C&M (Prof. Abeck), 2018.
[Ora-JEE] Oracle: JEE6 (JavaTM Platform, Enterprise Edition 6) API Specification – Index. *https://docs.oracle.com/javaee/6/api/index-all.html*



(1) A core part of the software development according to the Domain-Driven Design (DDD) is to make sure that the model and the implementation are the "same". A change of one of the two artifacts must lead to an equalizing change of the other artifact.
(1.1) The domain logic is only found in the domain logic layer. The other layers of the microservice architecture are separated from the domain logic as to promote reusability. The application logic ensures the access to the domain layer.

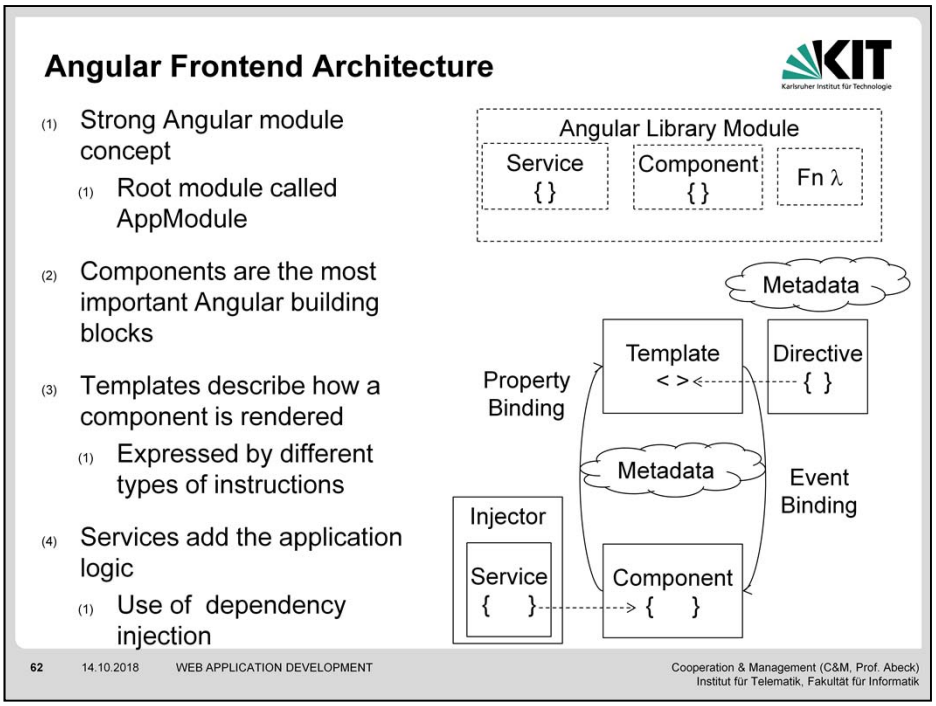
(<<shared entity>> TodoList …) TodoList is a central domain object of the TLM domain model as described on previous pages.

(43 public Todo createTodo ...) The method createTodo takes the parameter content that is required to create a new Todo.

(46 throwExceptionIfDuplicate ...) Two identical Todos in a TodoList are not allowed. Hence the new Todo is checked against duplicates.

(47 Todo todo = new Todo...) A new Todo is created and added to the todos list. Then the newly created Todo is returned.

[He18] Justin Hecht: Realisierung einer sensorbasierten Raumreservierung unter Einsatz der IoT-Middleware symbIoTe, Bachelor Thesis, Karlsruhe Institute of Technology (KIT), C&M (Prof. Abeck), 2018.



This page gives an overview of the modular architecture the Angular framework is based on.

(1) An Angular module is a class decorated with NgModule. Decorators (such as @NgModule) are functions that modify JavaScript classes.
(1.1) Each Angular App has at least one such Angular module class, the root module (conventionally called AppModule).
The Angular module system is complementary to the JavaScript module system in which each file is a module and all objects defined in the file belong to that JavaScript module. The two types of module systems must be well separated which is not that easy since both use the same vocabulary of "imports" and "exports".
(Library Module) A library module is a collection of JavaScript modules. Angular ships with several such library modules, called Angular libraries each beginning with the @angular prefix. Examples are: @angular/core, @angular/platform-browser.

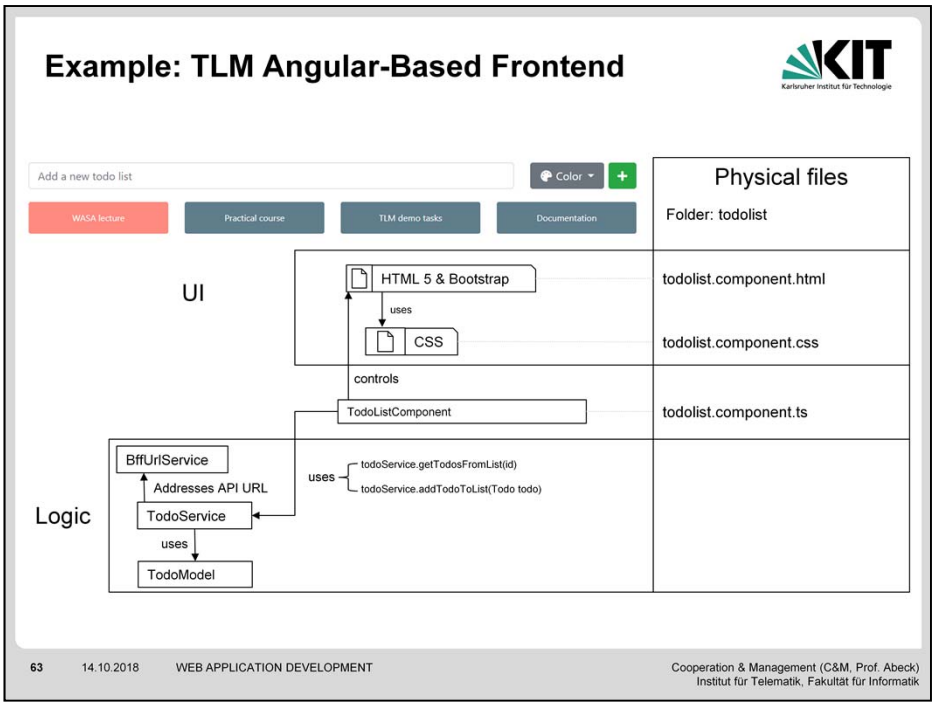
(Metadata) Describe how a class needs to be processed. They are inserted into classes with decorators.

(2) A component controls parts of the screen called view and each defined by a template. The component supports the view by providing presentation logic which is implemented as a class.

(3) A template is a form of HTML that tells Angular how to render the component.
(3.1) Directives give instructions on how templates should be rendered. There are three types of directives in Angular. First, there are components that are directives with templates. Second, there are structural directives, such as ngIf and ngFor, that manipulate the DOM layout by adding and removing DOM elements. There are also attribute directives, such as ngStyle, that change the appearance or behavior of elements, components or directives.
(Property Binding, Event Bindig) The mechanism for coordinating parts of a template and parts of a component is supported by the data binding concept of Angular. Data binding is conducted by adding markup bindings (specific forms) to the HTML template to tell Angular how to connect the component with the DOM.

(4) A service can be nearly everything, such as a logging service or data service. Or it can be a tax calculator or an application configuration.
There is nothing specifically Angular about services. Angular has no definition of a service. There is no service base class, and no place to register a service.
(4.1) (Injector) Angular uses dependency injection to provide new components with the services they need. When Angular creates a component, it first asks an injector for the services that the component requires.

[Ang-Arc] Angular: Architecture Overview. *https://angular.io/docs/ts/latest/guide/architecture.html*



The main structure of the Angular architecture is demonstrated with the example of the part of the TLM frontend which presents the data and functions related to a todo list.

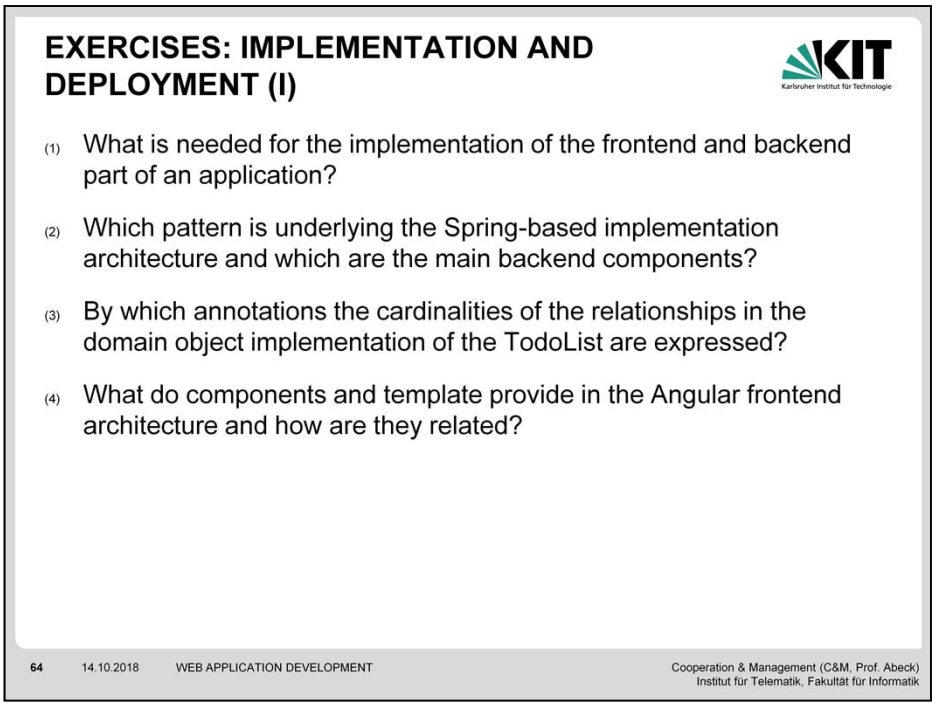
(TodoList) This component handles everything a todo list offers.

(HTML 5 & Bootstrap) The HTML defines which elements are shown in the browser. It includes Angular syntax like "*Ngfor" which is used for addressing each list element. The todolist.component.html uses the Bootstrap element "card".

(CSS) Elements can use custom style sheets. These are defines in the "*.css" classes.

(TodoListComponent) The file todolist.component.ts handles the actions from the HTML elements. For example, if a button on the web page is pressed, it is forwarded to this file. If an action requires a request, this is forwarded to the service.

(TodoService) The TodoService is the interface to the BFF. It forwards every request.

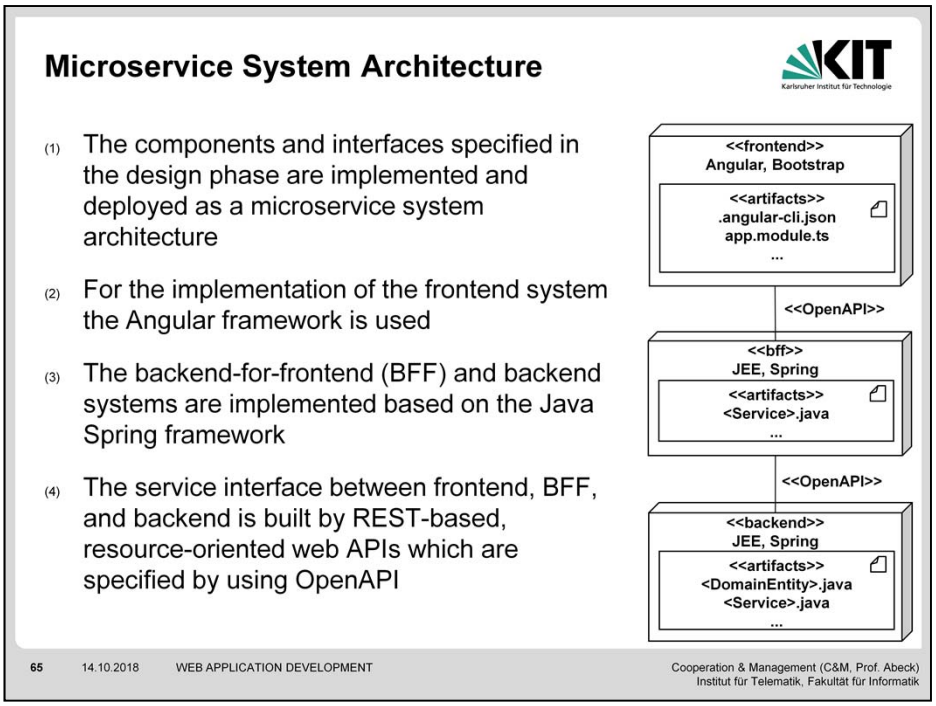


(1) ++Overview++
++Separation into Frontend and Backend Implementation++
- A prerequisite for the start of this implementation is the existence of the API specifications of the microservice architecture.

(2) ++Spring-Based Backend Architecture++
- The pattern is called MVC (Model-View-Controller).
- Backend components: BFF Controller, BFF Model, Microservice Controller, Microservice Model

(3) ++Example: TLM Domain Model and Implementation++
- @OneToMany: The class TodoList is related to many Todos since a todo list cant contain any number of todos.
- @ManyToOne: A user can have many, i.e. any number of todo lists.

(4) ++Architecture of an Angular Application++
- Component: controls parts of the screen called view which are defined by a template. The component supports the view by providing application/presentation logic which is implemented as a class.
- Template: is a form of HTML that tells Angular how to render the component.
- Relation: The mechanism for coordinating parts of a template and parts of a component is supported by the data binding concept of Angular. Data binding is conducted by adding markup bindings (specific forms) to the HTML template to tell Angular how to connect the component with the DOM (https://angular.io/guide/architecture#data-binding).



The microservice system architecture consists of a frontend, backend-for-frontend (BFF) and a backend. Angular is used as implementation technology for the frontend and Spring as implementation technology for the BFF and the backend microservices.

(1) A microservice architecture divides the software system into three distributed subsystem (or system parts) frontend, BFF, and backend. These systems communicate via resource-oriented web APIs.

(Diagram) The Unified Modeling Language (UML) provides the deployment diagram to model this physical view on the software application [MH06] as shown on the right hand side of the slide. The three-dimensional boxes represent the UML modeling element of a so-called node (which describes a computer system). On a node contain so-called artifacts are located which describe software programs running on that node.

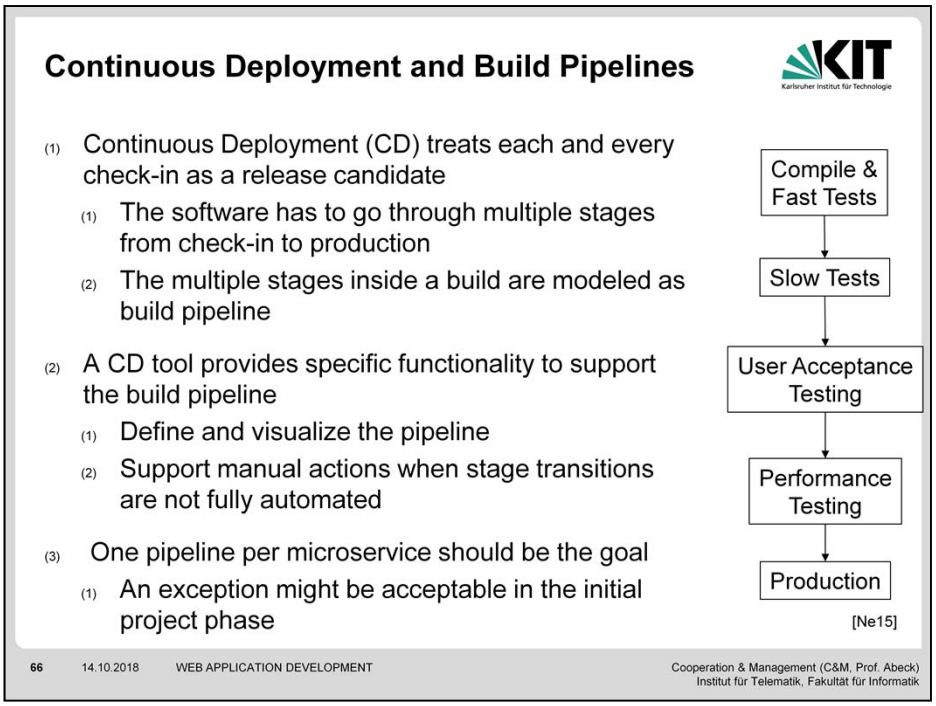
(2) Angular is a popular open-source framework to implement the frontend based on HTML (HyperText Markup Language) and Typescript which is an extension of JavaScript.
(.angular-cli.json) A JSON file (JavaScript Object Notation) which contains configuration information of the whole project setup.
(app.module.ts) A Transcript file which defines the modules that are bootstrapped by the framework at its start.

(3) Spring is one of the most popular open-source development framework for Java-based application servers. Many available open JavaEE technologies, such as Java Persistence API (JPA) are used and extended by Spring.

(4) The three system parts of the microservice architecture (usually) run on distributed client and server systems which communicate via the Web. The way how specification of the web APIs are specified is described in a previous chapter.

HTML	HyperText Markup Language
JPA	Java Persistence API
JSON	JavaScript Object Notation
UML	Unified Modeling Language

[MH06] Russ Miles, Kim Hamilton: Learning UM 2.0, O'Reilly, 2006.



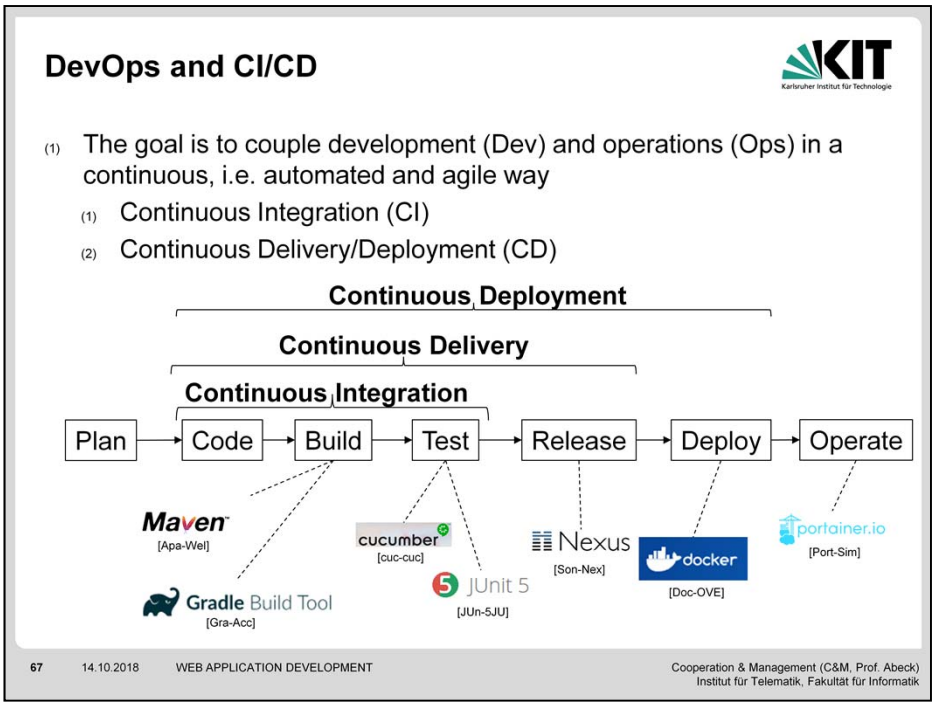
(1) Continuous Deployment (CD) gives constant feedback on the production readiness of each and every check-in [Ne15].
(1.1) The reason for the stages is that there is a sequence of the needed activities. An example is to conduct fast tests before slow tests (more general: to conduct different types of tests in a specific sequence).
(1.2) The figure on the right side shows the different stages of a build pipeline on an abstract level.

(2) The complexity of such a CD tool is often underestimated. It does not make sense to try to hack and extend Continuous Integration (CI) tools to make them do CD.
(2.1) The tool supports the developer to keep a good overview of the process and the current status.
(2.2) In the case of a manually executed "User Acceptance Test" (UAT) the tool supports this process by showing the next available build ready to be deployed in the UAT environment, deploy it and show the test result in the visualized build pipeline.

(3) The service is an artifact (coming with different, platform-specific shapes) that is created and moved through the pipeline.
(3.1) In the initial phase, the service boundaries are often not clear and changes across service boundaries are more likely. As soon as a service API stabilizes it should be moved into an own build.

CD	Continuous Deployment
CI	Continuous Integration
UAT	User Acceptance Test

[Ne15] Sam Newman: Building Microservices – Designing Fine-grained Systems. O'Reilly, 2015.

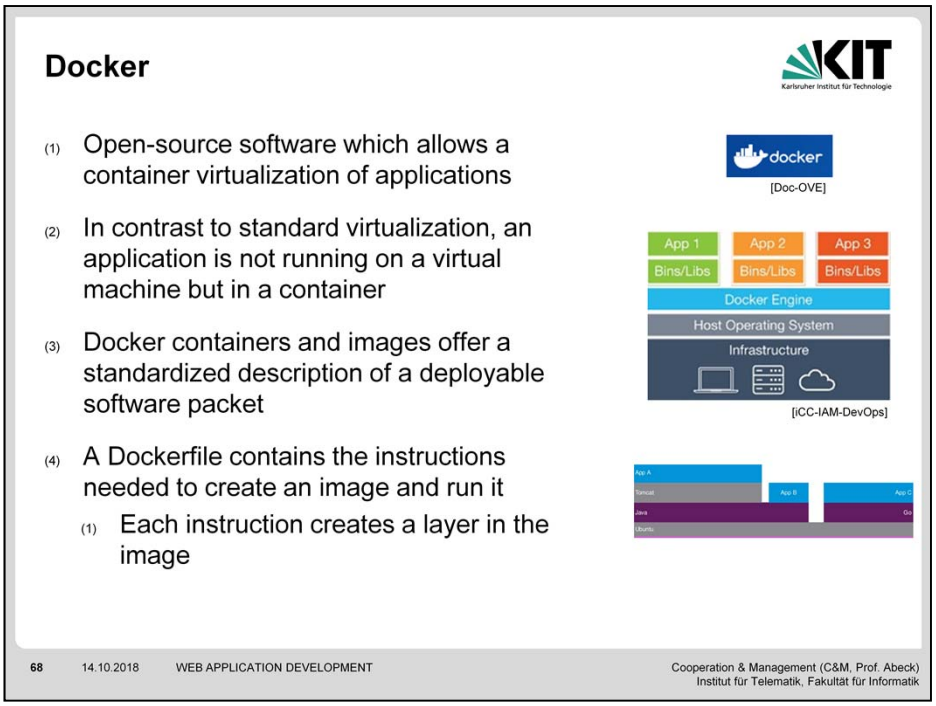


After the software has been implemented it must be built and deployed which requires another specific set of tools.

(1) DevOps contains the two terms development and operations. It summarizes all practices that bring together software development and software operations in a way that the development cycles are shortened and the deployment frequency is increased. Microservice architectures support the DevOps concept since microservices have the property that they can be tested and deployed independently from the whole software system. This is not the case for monolithic systems.
(1.1) Continuous integration (CI) is a DevOps practice by which the code from the involved developers are built and tested in short time intervals (e.g. several times a day) in order to discover and solve integration problems as soon as possible.
(1.2) Continuous delivery (CD) extends the CI practice by releasing the software after it was built and tested (i.e. continuously integrated).
(1.3) Continuous deployment (also abbreviated by CD) which is often confused with continuous delivery, means that every change is automatically deployed to production. Since the software must be released before it can be deployed, continuous deployment is an extension of the DevOps practice of continuous delivery.

(Maven) A Java tool by which a project's build, reporting and documentation can be managed.
(Gradle) A Java-based build management tool (comparable to Apache Maven) which uses a domain-specific language
(Cucumber) A library by which user acceptance tests based on Gherkin features can be carried out.
(Junit5) A framework to test Java programs based on unit tests.
(Nexus) A repository for build artifacts produced by Docker, Maven, npm, and others.
(Docker) A software system which supports container virtualization in order to deploy applications in isolated containers.
(Portainer) An open-source management user interface to manage a Docker host or swarm cluster.

[Apa-Wel] Apache: Welcome to Apache Maven. *https://maven.apache.org/*
[cuc-cuc] cucumber ltd: cucumber. *https://cucumber.io/*
[Doc-OVE] Docker Inc.: OVERVIEW. *https://www.docker.com/what-docker https://docs.docker.com/engine/docker-overview/*
[Gra-Acc] Gradle: Accelerate developer productivity. *https://gradle.org/*
[JUn-5JU] JUnit: 5 JUnit 5. *https://junit.org/junit5/*
[Son-Nex] Sonatype: Nexus Repository manager. *https://www.sonatype.com/nexus-repository-sonatype*
[Port-Sim] Portainer: Simple management UI for Docker. *https://portainer.io/*



Docker is a software product developed in Go by Docker Inc. The first version of Docker was published in 2013.

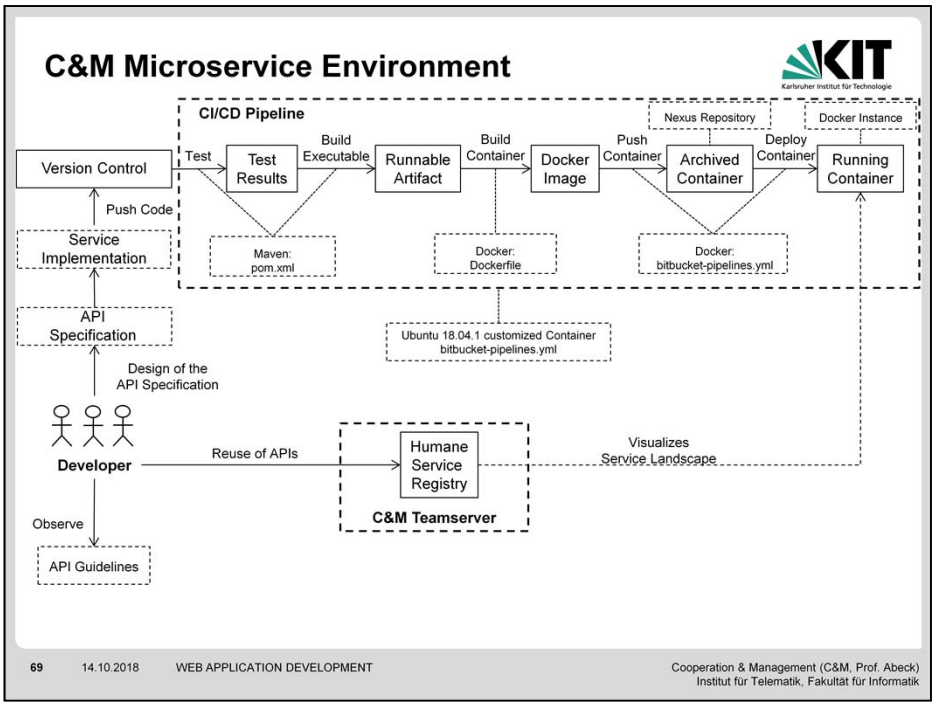
(1) Docker extends the existing container technology from Linux in a way that it provides a complete and easy-to-use solution for the creation and the operation of containers.

(2) Container virtualization offer applications the isolated usage of resources (CPU, RAM, network) without use of heavyweight virtual machines. Each container runs as an isolated process on the host operating system. This offers important advantages compared to virtual machine virtualizations, such as lightweight resource consumption (esp. CPU, storage), shorter start time, easier distribution.

(3) A container is a runnable instance of an image while an image is a read-only template with instructions for creating a Docker container. The standardized description leads to a portability of containers. In addition, deployment becomes easier and more stable.

(4) The instructions are formulated in a simple syntax. An example of a command that can be used in a Dockerfile is "docker run".
"$ docker run -i -t ubuntu /bin/bash" (i) creates a new container containing an ubuntu image, (ii) allocates a read-write filesystem as its final layer, (iii) creates a network interface including the assignment of an IP address to the container, (iv) starts the container and executes the /bin/bash command.
(4.1) One important property of layers is that they are shared between all installed images.
In the example, the (operating system) layer "Ubuntu" and the (programming language) layer Java are shared by the applications A,B, and C.

[Ak17] Ali Akil: Einführung und Erprobung der Virtualisierungstechnik Docker beim IT-Dienstleister ATIS, Karlsruher Institut für Technologie, Cooperation & Management (Prof. Abeck), Bachelorarbeit. *https://team.kit.edu/sites/cm-tm/Archiv/2017/BA_Akil*
[CM-W-MIC] Cooperation & Management: MICROSERVICES, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*
[iCC-IAM-DevOps] Daniel Deckers: IAM in einer devOps-Welt, Autor: Daniel Deckers. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*
[Doc-OVE] Docker Inc.: OVERVIEW. *https://www.docker.com/what-docker https://docs.docker.com/engine/docker-overview/*



This page illustrates an example of a microservice environment as it is established at C&M.

(Humane Service Registry) This registry is used during the development to guarantee a consistent design of the architecture. The humane service registry visualizes the service landscape by offering a user-defined list which is stored in the C&M Teamserver.

(Version Control) The source code of the services is organized with the version control system Git and stored in the Bitbucket environment.

(CI/CD Pipeline) The CI/CD pipeline is a customized pipeline and it is executed whenever code is pushed to the repository. Hereby a step initiates the next one as long as no error occurs. Each step runs in its own Linux Docker container. The container is defined by the file "bitbucket-pipelines.yml" which is stored in the version control system.

(Test Results) The test results are produced by running "maven verify" [Mav-Int]. This includes a validation whether all information, like dependencies, are available, compile the source code and to test the code. The project information and dependencies which are needed are given by the "pom.xml".

(Runnable Artifact) Furthermore "maven verify" packages the compiled code to an executable. In our case it is a lightweight jar file. This jar is then stored as a pipeline artifact to be used in the next step.

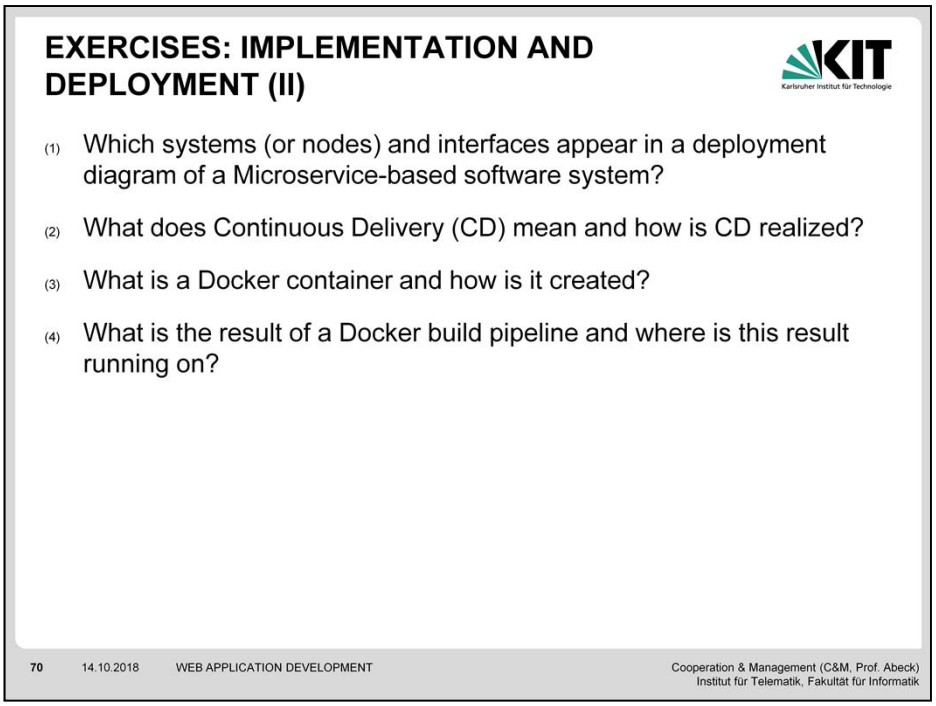
(Docker Image) The Docker image is a ready to run container to be deployed in Docker. Therefore, information about the execution parameter for the jar, its location and exposed ports are needed. These and other parameters are stored in the Dockerfile.

(Archived Container) After the creation of the Docker image the image is pushed to the Nexus repository which is similar to a version control system for build artifacts. Information about the repository's location are given by the "bitbucket-pipelines.yml".

(Running Container) Finally, the container previously running is removed from the Docker host and the new one is deployed from the repository. Therefore, the location of the Docker host and start parameters (e.g. port mappings) are provided by the "bitbucket-pipelins.yml". Furthermore, any version of another container can be deployed from the repository. This might be necessary if there are certain dependencies.

CD	Continuous Deployment
CI	Continuous Integration

[Mav-Int] Maven: Introduction to the Build Lifecycle. *https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html*



(1) ++Deployment Diagram of the Implementation Architecture++
-Nodes: Fontend, BFF, Backend system
- Interfaces: Web APIs, e.g. specified by using OpenAPI

(2) ++Continuous Delivery (CD) and Build Pipelines++
- CD treats each and every check-in as a release candidate
- CD requires a build pipeline which consists of several stages (resulting from different fast/slow/user acceptance/performance tests) which the software must pass.

(3) ++Docker++
- A Docker container runs as an isolated process on the host operating system and provides a virtualized environment to run applications.
- A Docker container is a runnable instance of an image.
- Created by instructions in a Docker file

(4) ++C&M Microservice Environment++
- The result is a application container image (Docker image) which runs in a container (Docker container).

API	Application Programming Interface
BDD	Behavior-Driven Development
BFF	Backend For Frontend
CD	Continuous Delivery
CI	Continuous Integration
COAP	Constrained Application Protocol
CRUD	Create, Read, Update, Delete
CVS	Concurrent Versions System
C&M	Cooperation & Management
DB	Data Base
DDD	Domain-Driven Design
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
JPA	Java Persistence API
KIT	Karlsruhe Institute of Technology
MVC	Model View Controller
MVP	Minimum Viable Product
POJO	Plain Old Java Object
RAML	RESTful API Modeling Language
RCS	Revision Control System
REST	REpresentational State Transfer
RM	Resource Model
SemVer	Semantic Versioning
TLM	TodoListManagement
UAT	User Acceptance Test
UML	Unified Modeling Language
URI	Universal Resource Identification
VCS	Version Control System
XML	eXtensible Markup Language

[Ak17] Ali Akil: Einführung und Erprobung der Virtualisierungstechnik Docker beim IT-Dienstleister ATIS, Karlsruher Institut für Technologie, Cooperation & Management (Prof. Abeck), Bachelorarbeit. *https://team.kit.edu/sites/cm-tm/Archiv/2017/BA_Akil*

[Ang-Arc] Angular: Architecture Overview. *https://angular.io/docs/ts/latest/guide/architecture.html*

[Atl-Bec] Atlassian: Become a git guru. *https://www.atlassian.com/git/tutorials*

[Atl-Cre] Atlassian: Create Your Account. *https://bitbucket.org/account/signup/*

[Atl-Get] Atlassian: Getting Started. *https://www.atlassian.com/git/tutorials/setting-up-a-repository*

[Atl-Git] Atlassian: Gitflow Workflow, Git Tutorial. *https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow*

[Atl-Lea] Atlassian: Learn Git with Bitbucket Cloud. *https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud*

[Atl-Rep] Atlassian: Repositories. *https://confluence.atlassian.com/bitbucket/repositories-675385631.html*
https://ic-consult.atlassian.net/wiki/spaces/KK/pages/121602185/Durchf+hrung+von+Entwicklungsarbeiten

[Atl-Wha] Atlassian: What is version control. *https://www.atlassian.com/git/tutorials/what-is-version-control*

[Ca01] David Carrington: Software Engineering Tools and Methods, IEEE – Trial Version 1.00, 2001. *https://pdfs.semanticscholar.org/100c/1b90a8870dd256ceb98c6a831bba3dc9cf92.pdf, https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*

[CM-C-Dur] Cooperation & Management: Durchführung von Entwicklungsarbeiten, Confluence-Seiten.
https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/CM-Dokumente

[CM-W-BEH] Cooperation & Management: BEHAVIOR-DRIVEN DEVELOPMENT, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*

[CM-W-DOM] Cooperation & Management: DOMAIN MODELING, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*

[CM-W-TLM] Cooperation & Management: TLM PRACTICAL COURSE WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-2.Leitfäden*

[CM-W-UNI] Cooperation & Management: UNIFIED MODELING LANGUAGE, WASA-Kurseinheit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*

[CM-W-WEB] Cooperation & Management: WEB APPLICATION DEVELOPMENT, WASA Course Unit. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-1.WASA*

[CMU-SA] CMU: Software Architecture Getting Started Glossary Modern Software Architecture Definitions. *https://www.sei.cmu.edu/architecture/start/glossary/moderndefs.cfm*

[CS14] Scott Chancon, Ben Straub: Pro Git, Apress, Second Edition, 2014. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*

[Ev04] Eric Evans: Domain-Driven Design – Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.

[Gi18] Pascal Giessler: Domänengetriebener Entwurf von ressourcenorientierten Microservices, Dissertation, Karlsruhe Institute of Technology (KIT), Cooeration & Management (C&M, Prof. Abeck), 2018. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-5.Publikationen/1.Vergangene_Jahre/1.Promotionen*

[HP16] Valentin Haenel, Julius Plenz: Git - Verteilte Versionskontrolle für Code und Dokumente, 2016. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/Software-Entwicklung*

[Hü18] Tobias Hülsken: Eine Microservice-Architektur für das Identitäts- und Zugriffsmanagement, Masterarbeit, Karlsruher Institut für Technologie (KIT), C&M (Prof. Abeck), 2018.

[KR05] James F. Kurose, Keith W. Ross: Computer Networking – A Top-Down Approach Featuring the Internet, Pearson Education, 2005.

[MH06] Russ Miles, Kim Hamilton: Learning UM 2.0, O'Reilly, 2006.

[Ne15] Sam Newman: Building Microservices – Designing Fine-grained Systems. O'Reilly, 2015.

[OAI-Ope] OAI: OpenAPI Specification Version 3.0. Website, 2017. – URL: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md

[Ot13] Otto: Architekturprinzipien – dev.otto. *https://dev.otto.de/2013/04/14/architekturprinzipien-2/*

[RH08] Ralf Reussner, Wilhelm Hasselbring: Handbuch der Software-Architektur, dpunkt Verlag, 2008.

[RM+06] John Reekie, R. J. Mcadam, Rohan McAdam: A Software Architecture Primer, Angophora Press, 2006.

[RW+15] Seb Rose, Matt Wynne, Aslak Hellesoy: The Cucumber For Java Book. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD*

[Sm15] John Ferguson Smart: BDD in Action – Behavior-Driven Development for the whole software lifecycle. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD*

[Ta06] Andrew .S. Tanenbaum: Computer Networks, Prentice Hall, 2006.

[tut-Jir] tutorialspoint: Jira. *https://www.tutorialspoint.com/jira/*

[VA+08] Oliver Vogel, Ingo Arnold, Arif Chughtai, Edmung Ihler, Timo Kehrer, Uwe Helog, Uwe Zdun: Software-Architektur: Grundlagen – Konzepte –Praxis, page 48ff. Springer Verlag, 2008.

[WH12] Matt Wynne, Aslak Hellesoy: The Cucumber Book. *https://team.kit.edu/sites/cm-tm/Mitglieder/2-4.Literatur/BDD*