

Software Engineering for Business Applications

Lecture Notes

Efe Kamasoglu

February 8, 2023

1 IT Support for Business Applications

1.1 Classification of Business Applications

- **Definition "Business Application":**
 - in narrower sense: totality of all programs, i.e. **application software**, and associated **data** for a concrete business use case
 - in broader sense: additionally **hardware**, **system software** and necessary **communication** facilities required for the use of application software
- **Two roles of Business Applications:**
 - **supporting, improving** or **automating** existing operational processes in bookkeeping, accounting, etc. (size, speed, correctness...)
 - **enabling** new products and services (e.g. online shopping and banking)
- **Classification of Business Applications by Business Purpose:**



Examples of

- **administrative systems:** financial accounting, payroll accounting, administration of stocks
 - **disposition systems:** calculation and cost accounting, material procurement, field service control
 - **management information systems (MIS):** use of internal company data, use of external data, combination of multiple data sources in a flexible form
 - **planning systems:** planning of individual functional areas, integrated planning of several functional areas, corporate planning
- **Cross-Cutting Applications:**
 - independent of company hierarchy and functional domains
 - used either directly via user interface or programmatically via administration and disposition systems
 - *Examples:* office suites, groupware, workflow management systems

- **Enterprise Resource Planning (ERP): ERP system** is an integrated business application (suite, collection of programs), which supports all essential functions of administration, disposition and management with a common interface and a shared and integrated data management.

 - consists of platform and function-oriented application components that exchange info and events
 - is realized as (customizable) standard software
 - *Examples:* external accounting, controlling, procurement
 - Today's ERP systems support an **extended value chain**¹.

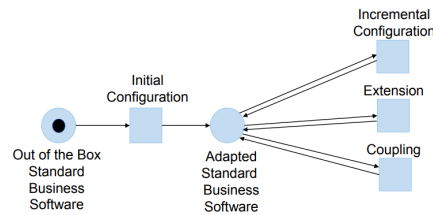
1.2 Standard and Custom Software

- **Standard Software vs. Custom Software:**
 - **Standard software** (*e.g. SAP*)
 - * developed for specific **market**
 - * distributed by a software house
 - * can be used by **several companies**
 - * implements "standard business processes" at its core
 - * maintained by **manufacturer**, adapted to changes
 - * must or can be **customized** to company (e.g. authorizations and roles, currencies)
 - **Custom software**
 - * specifically developed for **one company**
 - * tailored to specific business processes/requirements
 - * result of a project for a known client
 - * **individually** maintained and adapted to changes

¹**Value chain** is a business model that describes the full range of activities needed to create a product or service.

- **Adaptation Techniques for Standard Business Software:**

- Adaptation of operational standard software can be divided into **Configuration, Extension and Coupling (= Customizing)**.



- **Configuration** describes functionalities and techniques
 - * that are obligatory on first deployment
 - * that allow to define predefined settings
 - * that lead to an individual variation of standard software
- **Extension** describes functionalities and techniques
 - * that are optional for productive use
 - * that allow to map requirements not foreseen by manufacturer
 - * implemented by manufacturer to expand the range of services
- **Coupling** refers to functionalities and techniques
 - * to connect external systems of other manufacturers
 - * to connect external systems of the same type
 - * that are predefined in the form of data file formats, APIs, or communication protocols
- *Example:* mapping the structure of a company to SAP applications via organizational units (can be assigned to single or multiple apps)

- **Configuration: Challenges**

- A **standard software** must
 - * provide all relevant configuration options
 - * support a wide range of different corporate structures and processes
 - * check dependencies between these many variants
 - * provide appropriate documentation about the effects of individual configurations
- **Consequences:**
 - * need for experts who are familiar with configuration options of each release and component
 - * scarcity of such experts

- * expensive training
 - * expensive consultancy services
- **Examples for Extensions:**
 - automation of **multi-step business workflows**
 - integration of company-specific calculations/rules/checks
 - connecting customers
- **Coupling Options:**
 - different coupling options depending on the scenario
 - programming language used for coupling
 - available mechanisms to couple
- **Multi Tenancy:** Software multitenancy is a software architecture in which a single instance of software runs on a server and serves multiple tenants (e.g. companies).
 - several companies can be represented in one system
 - distinction between tenant-dependent and -independent data
 - supporting tenant-dependent authorization (e.g. A may only perform transactions in client 002)
 - individual adaptations of tenants (e.g. currency, couplings)
- **Multilingualism:**
 - **Multilingualism of a business information system** makes it possible to
 - * store and display texts in different languages in the system
 - * assigning graphics and symbols specific to different languages
 - Multilingualism requires
 - * that one system can process all relevant character sets at once
 - * storage and recognition of words, numbers etc.
 - * that a system can assign users to languages or user can choose their own
 - * that texts (graphics, symbols) can be assigned to a language
- **Localization (l10n):** Adaptation of a software product to meet the language, culture, and other requirements of each locale (e.g. adaptation of graphics, currencies, date and time)
- **Internationalization (i18n):** Process of preparing a software-based product for localization (to support global markets)

1.3 Characteristics of Business Applications

- **Multiple Stakeholders and changing requirements:**
 - **Requirements Elicitation and Requirements Management**
 - * many stakeholders, different views and concerns
 - * Waterfall: upfront requirements document and/or technical specification => Req. Documentation
 - * Issue: changing requirements once IT support is implemented
 - * Agile: incremental and iterative => Agile Req. Engineering
 - * typically, very large number of requirements
 - * need for formalization and early consistency checking => Conceptual Modeling
 - * need for cost and time prediction => Software Estimation
 - **Programming Challenges**
 - * design, implement and test changes in an existing complex system => Change Mgmt.
 - * deliver incremental changes without invalidating existing data => Release Mgmt.
 - * parallel development at manufacturer and at customer site => Version Mgmt.
 - * automated and quality-controlled assembly of application software => Build Mgmt.
- **Persistent Data and Concurrent Data Modification:**
 - **Data consistency** is a must:
 - * many users perform **transactions** simultaneously on central databases
 - * data must not be lost even in case of system failures.
 - **Programming challenges:**
 - * database is managed by an independent application, on a different server / hardware
 - * object orientation is not supported by common data bases
 - * database concepts must be transferred to the application logic (transactions, rights, primary keys)
- **Distributed Actors and Data Repositories:**
 - **Many users access central data concurrently:**
 - * users need data in different locations at different times
 - * Client-Server architecture => Layered Architectures
 - * web clients => REST protocol
 - **Programming challenges:**

- * software components must be able to find in network => Naming services
- * communication always via a network => Serialization² & failed execution
- * authentication and authorization => Security
- * concurrent accesses => Transactions
- **Integration of Data and Application from (Semi-)Autonomous Sources:**
 - **Separation of applications and data repositories:**
 - * multiple apps work on independent or shared data resources
 - * multiple apps communicate with each other => RPC, Message Passing
 - * business processes involve multiple apps => Workflow Mgmt. Systems
 - * application landscapes with lots of interacting applications => Enterprise Architecture Mgmt.
 - **Programming challenges:**
 - * integration of multiple languages and databases
 - * loose coupling through interfaces to avoid code change propagation
 - * error recovery to avoid runtime failure propagation
- **Scalability:**
 - **Growing number of users and data volume**
 - * business apps are used by thousands of employees world-wide around the clock
 - * customers and business partners interact directly with business apps and expect real-time sub-second response times
 - * volatile load (e.g. online shop in christmas season vs. summer season)
 - **Programming challenges:**
 - * delayed execution of resource-intensive operations => Batch processing³
 - * dynamically increasing/decreasing number of users => Instance pools
 - * single server cannot handle the load => Load balancing, Caching

²**Serialization** is the process of translating a data structure into a format that can be stored or transmitted and reconstructed later.

³**Batch processing** is when a computer processes a number of tasks that it has collected in a group. It is designed to be a completely automated process, without human intervention.

2 Requirements Engineering

- **Software requirements** express the needs and constraints placed on a software product.
- **Requirements engineering** is concerned with **elicitation**, **analysis**, **specification** and **validation** of software requirements as well as the management of requirements.
- **Requirements Management** deals with the administration and maintenance of requirements documents, in particular:
 - change requirements (change management)
 - trace and link requirements (requirements tracing)
 - verify requirements

2.1 Traditional Requirements Engineering

- **Objectives of Requirements Management:**
 - **Efficient** preparation of **high quality** requirements and system specifications,
 - * coordinated with all stakeholders (different objectives and interests)
 - * coordinated with all specifications and constraints
 - * evaluated according to profitability and feasibility
 - **Specification documents** are basis for:
 - * contract negotiation and contractual agreements
 - * coordination between the stakeholders (customers, developers)
 - * design, realization, integration
 - * software acceptance (test specification)
 - * future developments, projects
- **Requirement Classification:** Distinction between functional and non-functional requirements and constraints:
 - **Functional requirements** describe interactions between the system and its environment independent of their realization.
 - **Non-functional requirements** describe general properties of the system.
 - **Restrictions (Constraints)** determine the solution space for the realization.
- **Stakeholder Management:** It includes

- processes required to identify people that could impact or be impacted by the project
- to analyze stakeholder expectations and their impact on the project
- to develop appropriate management strategies for effectively engaging stakeholders in project decisions and execution
- **Requirement Specification:**
 - technical result document of requirement identification phase
 - **contains** stakeholder identification, functional and non-functional requirements, constraints, evaluation plan and metrics
 - list of all deliverables and services to be fulfilled by contractor within contract as defined by customer
 - **what** is to expect from the solution (product)
 - formulation of requirements should be as general as possible and as restrictive as necessary
 - enables the contractor to develop optimal solutions
- **Requirements Validation: Validation, Consistency check** (no conflicts), **Completeness check, Reality check, Verifiability**
- **Functional Specification:**
 - defines the purpose of the system
 - solution proposal created by contractor based on the requirement specification provided by client
 - **contains** target determination, product usage, environment (e.g. hardware), functions, UI, global test cases
 - system description or solution specification, which describes **how** the solutions is to be realized (concrete solution approaches)
 - the **what** from **requirement specification** is detailed

2.2 Agile Requirements Engineering

- **Requirements Engineering and Agile Software Development:**
 - **Agile software development** focuses more on **continuous collaboration** (workshops, interviews etc.) with stakeholders instead of relying on **specification documents** (*example: SCRUM*)
 - **Traditional requirements engineering**
 - * focuses on customer collaboration mainly at an early phase of the project (longer change cycles)
 - * emphasizes a heavy-weight process with extensive, **static specification documents**

- **Agile requirements engineering**
 - * fosters communication with the customer during the whole development process to **continuously update requirements**
 - * focuses less on extensive documentation, but specification documents **might be necessary** because of legal or contracting reasons etc.
 - * includes activities and artifacts that are similar to classical requirements engineering activities
- **Typical Requirement Artifacts in Agile Software Development:**
 - user story, story card, use case, scenario, UML diagram, prototype
- **User Stories:**
 - explanation of a software feature written from the perspective of the end user
 - most frequently used artifact in **agile software development**
 - mnemonic for writing good user stories: INVEST⁴
- **Typical Requirements Engineering Challenges:**
 - different interest groups can raise **conflicting requirements**
 - the people who **pay** for the system are rarely the ones who **use** it
 - the organization and the technical environment may **change** after the system rollout
 - requirements that change during implementation (Change Requests) can lead to additional costs -> project duration/milestones can be affected significantly

3 Conceptual Modeling with UML

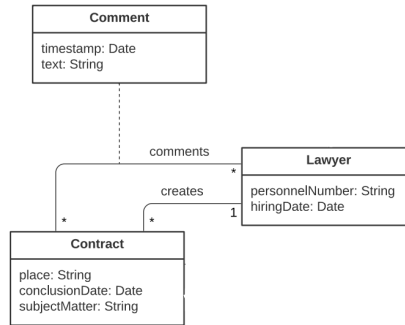
- **Conceptual Class Diagram vs. Implementation-Oriented Diagram:**

| | Conceptual | Implementation-Oriented |
|------------------------------|------------|-------------------------|
| Visibility (private, public) | No | Yes |
| Attributes with data types | Yes | Yes |
| Methods | No | Yes |
| Generalization / Inheritance | Sparingly | If useful / meaningful |
| Abstract classes | No | If useful / meaningful |
| Association classes | Yes | No (resolved) |

- **Associations between Classes:**

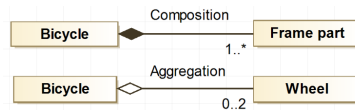
⁴independent, negotiable, valuable, estimable, small, testable

– **Multiplicity:**



A *Lawyer* can create **multiple** *Contracts*, whereas every *Contract* has a **single** *Lawyer*. -> creates (action) on the side of Lawyer (actor)

- **Aggregation:** implies a relationship where the child can exist independently of the parent (part of the parent)
- **Composition:** implies a relationship where the child cannot exist independent of the parent
- Example:



4 Software Estimation

4.1 Fundamentals of Estimation Methods

- **Software Estimation:**
 - In principle, software estimation relies on **forecasting effort**, from which cost and duration are derived.
 - Regardless of the project and software methodology applied, every initiative requires the definition of a **budget** and a specific **time frame** necessary to deliver a final outcome.
 - These two are obtained during the **early stages** of the project life-cycle through the process of estimation.

- **Estimation** aims to provide an **approximation** of the amount of resources required to complete project activities and produce a product or service in accordance to specified **functional** and **non-functional characteristics**.
- **Software estimation conducted in early phases of the project lifecycle:**
 - * necessary for contract negotiations
 - * predict expected efforts (and derived costs) for a software project before implementation
 - * best possible estimation given the available info
- **Agile estimation:**
 - * estimation of individual requirements during project
 - * incremental allocation of developers in the most efficient manner
 - * cost estimates are made several times during development project with varying degrees of detail
- **Software Estimation: Cone of Uncertainty**
 - At the beginning of the project, not much is known about the product/project -> estimates underly high uncertainty
 - As the project progresses, more information is available -> decrease in uncertainty
- **Software Estimation: Costs**
 - **Cost categories:**
 - * **Development costs:** costs to produce a software product
 - * **Personnel costs:** major share of development costs for personnel
 - usually low costs for office materials etc. in relation to the personnel costs
 - proportionate allocation of CASE⁵ environment costs (including hardware and software) for product development

4.2 Traditional Software Estimation

- **Sneed's Devil's Square:**
 - Quantity
 - Quality
 - Development duration
 - Cost

⁵Computer power-assisted software package Engineering

are mutually dependent.

- **Quantity:**
 - size of program code (example basis of assesment: LOC⁶)
 - functional and data scope
 - possible additional weighting with complexity
- **Quality:**
 - higher quality requirements => greater effort
 - no **THE quality**, but different quality characteristics
- **Productivity:**
 - influenced by many different factors
 - number of communication links grows **quadratically** with the team size
- **Development time:**
 - need more members to shorten development time
 - more members => more communication effort
 - higher communication => decrease in productivity
- **Methods for Effort Estimation:**
 - **Estimation Strategies:**
 - * **Top-Down:** estimation of the total project effort using mathematical algorithms based on the functional requirements
 - * **Bottom-Up:** expenses for each expense item are calculated separately and added to calculate the total project effort
 - **Comparison methods:**
 - * estimation based on effort analysis of already accomplished similar developments
 - **Algorithmic methods:**
 - * effort calculated with algorithmic methods
 - * based on statistical models or actual expenditure of already completed projects
 - **Key figure methods:**
 - * total cost of the software product determined by estimating the cost of individual units or project phases
 - None of the listed basic methods alone is sufficient.

⁶Lines of Code

- Depending on the point in time and knowledge of effort-relative data, one or the other method should be used.
- **Concrete Procedures for Effort Estimation:**
 - **Goal:** Combine advantages of several effort estimation methods to deliver accurate results. (*example: Function Point Method*)
- **Function Point Method:** It is a combined relation and weighting method.
 1. **Categorization** of each product requirement (input, query, output, database, reference data)
 - **Input:** by the user
 - **Output:** displaying query results, calculated data
 - **Query:** performed on the **database** of the system, read and write
 - **Reference data:** used to validate input, generate the output or construct the query (read-only)
 2. **Classification** of each product requirement
 - **simple**
 - **medium**
 - **complex**
 3. **Entry** into calculation form
 4. **Evaluation** of influencing factors
 - the **influence factors** refer to the application as a whole and not to individual functions or function points
 5. **Calculation** of the evaluated Function Points (FP)
 6. **Determination** of the personnel expenses based on a FP-PM curve or table
 - significant productivity decreases in large projects (FP-PM⁷: increase in FP => increase in PM) -> non-linear growth
 7. **Update** of empirical data as an estimation basis for follow-up project
 - After completion of a development estimated with the Function Point Method, the new value pair (FP, Actual PM) is used to update the existing curve.
- **Function Point Method: Requirements:**
 - evaluation once the project requirements are known
 - evaluation by employees with sufficient knowledge of requirements
 - product considered from the perspective of client

⁷**FP:** function points, **PM:** person month (= MM: man month)

- company-specific training, guidelines are needed to minimize the effect of subjective individual estimates during the classification and evaluation of influencing factors
- actual efforts must be measured for post-calculation
- **Function Point Method: Advantages**
 - product requirements, not LOC as starting point
 - adaptability to different application areas (change of categories)
 - adaptability to new techniques (change of influencing factors, influence evaluation)
 - adaptability to company-specific environments (if, ie and class factors per class)
 - refinement of the estimate according to the development process
 - first estimate is possible at a very early stage (planning phase)
 - good estimation accuracy
- **Function Point Method: Disadvantages**
 - only total effort can be estimated -> conversion to individual phases must be made using a percentage-based method
 - personnel-intensive, not easy to automate
 - too strongly function-oriented
 - influence factors do not clearly separate project and product characteristics

4.3 Agile Estimation Methods

- **Estimation in the SCRUM Framework:**
 1. Estimation of **Story Points**⁸ for each item in the **Product Backlog**
 - an **ordered list** of everything that is known to be needed in the product
 - **Product Backlog Refinement:** act of adding detail, **estimates**, and order to items in the **Product Backlog**
 - **User Story** is the unit with which software features are **estimated** and developed.
 2. **Time Estimation** (in days) for each item in the **Sprint Backlog**
- **Estimation with the help of Planning Poker:**

⁸**Story points** are units of measure for expressing an estimate of the overall effort required to fully implement a product backlog item or any other piece of work.

- reason to use **planning poker** is to **avoid the influence of the other participants** (group thinking)
- estimates are **story points** from different members (developers)
- estimates are revealed simultaneously to assure the indepenence between group members
- estimates are used during **release** and **sprint planning** meetings to create release and sprint plans