Studierendenprojekt 3. Semester

Von der Aufgabenstellung zur Projektplanung II

Dominique Brodbeck
dominique.brodbeck@fhnw.ch

Part I:

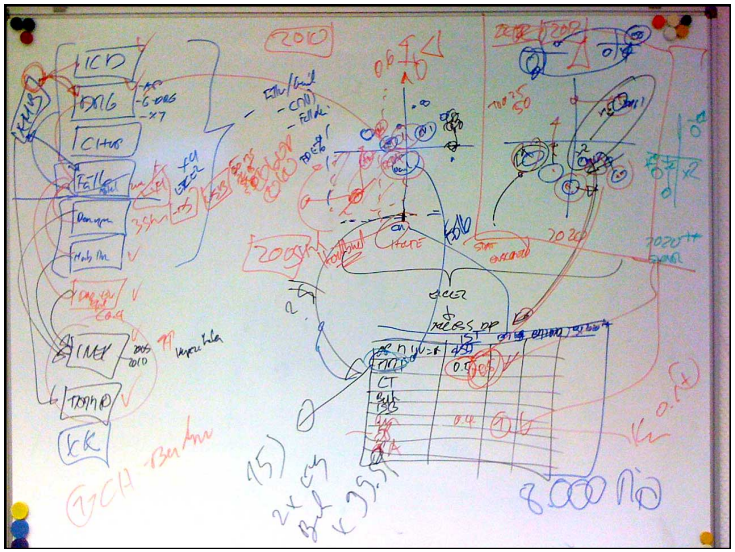
The Software Development Process

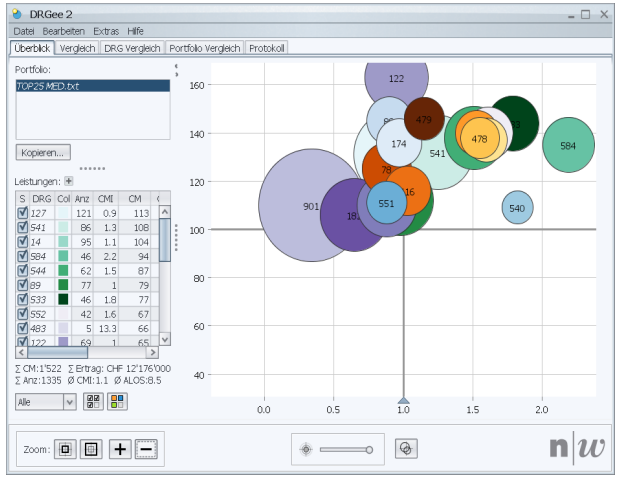
Part II:

Requirements

Part III:

Use Cases





105 Klassen

9956 Zeilen Java Code

"Do requirements, then design, then build" does not work for software

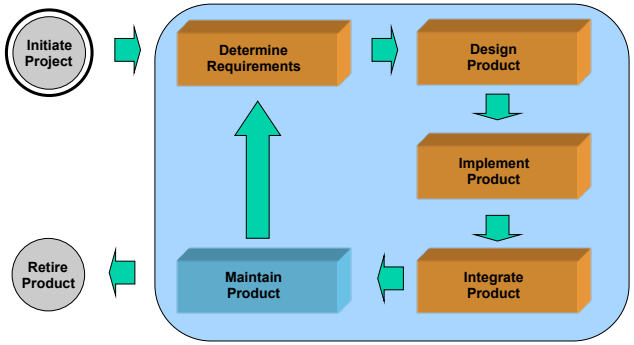
- Software is far more complex than hardware
- General Opinion: "It's easy to change software"
- Software is invisible and hard to visualize
  - Complete views - incomprehensible
  - Partial views – misleading
- Requirements are unstable

Requirements are unstable because of…

- …the unparalleled flexibility and options available for software;
- …the difficulty to fully, accurately, and appropriately speculatively define a software system (without feedback-adaptation cycles);
- …fast-changing market forces, which motivate changes

The Software Process

- A (very rough) Overview



Activities produce artifacts

- Domain Model
- Use Case Model
- Software Quality Assurance Plan - SQAP
- Software Configuration Management Plan - SCMP
- Software Project Management Plan - SPMP
- Software Requirements Specification - SRS
- Software Design Document - SDD
- Source Code Documentation
- Software Test Document
- User Manual
- Installation Guide

Activity: Determine Requirements

- Goal
  - Knowing what client needs, not what client wants
  - Make requirements „implementable“
- Activity
  - Interview, ask, search for requirements
  - Analyze requirements and problem domain
- Artifacts
  - Record of discussions and observations
  - Scenarios, user stories
  - Rapid prototypes
  - Software Requirements Specification (SRS)

Activity: Design

- Goal
  - Software is designed for implementation
- Activity
  - Develop appropriate software design
    - System design
    - Architectural design
    - Detailed design
- Artifacts
  - Architecture documents
  - Detailed design

Activity: Implementation

- Goal
  - Correctly runnable components
- Activity
  - Coding, testing
- Artifacts
  - Source code (with comments)
  - Documentation generator
  - Test cases

Activity: Integration

- Goal
  - Integrated overall system for productive use
- Activity
  - Combine components and modules
  - Testing
- Artifacts
  - Commented source code
  - Manuals
  - Test Cases

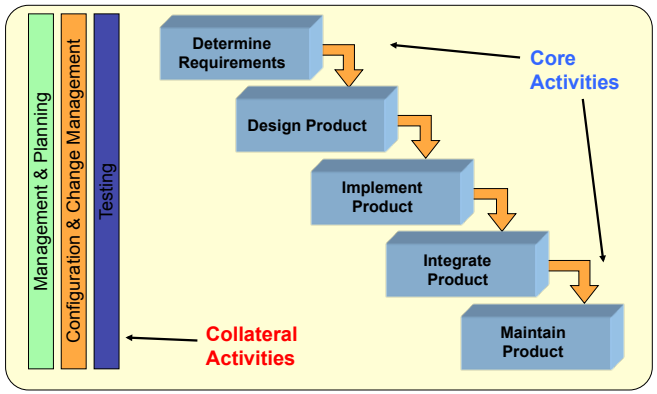
Activity: Maintenance

- Goal
  - Software adapted to changing environment
- Activity
  - Fix bugs
  - Refactor
  - Migrate
- Artifacts
  - Same as in development activities

Activity: Plan Project

- Goal
  - Managed and controlled process
- Activity
  - Plan and manage development process
  - Control schedule and budget
  - Plan configuration management
  - Plan quality assurance
- Artifacts
  - Software Configuration Management Plan
  - Software Quality Assurance Plan
  - Project Plan

The activities need to be organized

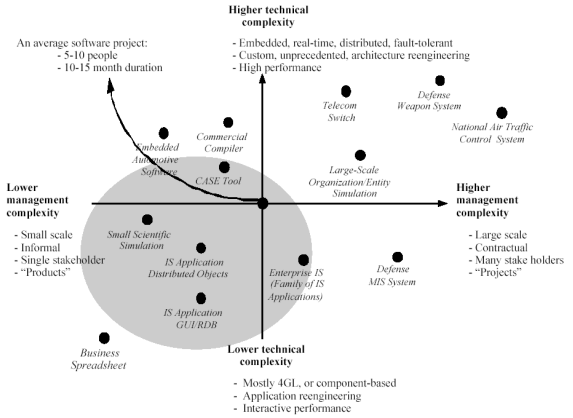


Life Cycle Models

The way **how** the software process activities are executed

- Different Approaches
  - Sequential vs. iterative & incremental (or predictive vs. adaptive)
  - heavyweight vs. lightweight

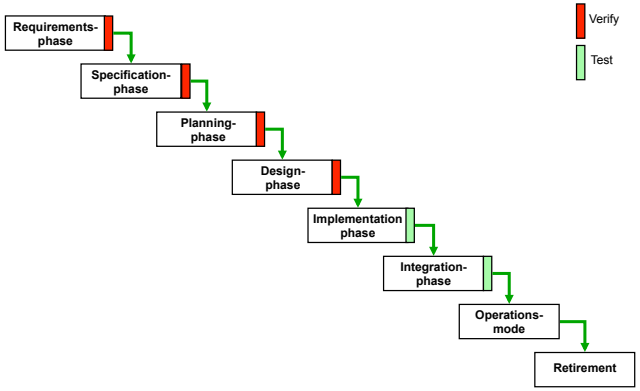
Software Complexity



Life Cycle Models

- Sequential
  - Waterfall Model
  - V-Model
  - ...
- Iterative & Incremental
  - Unified Process
  - eXtreme Programming (XP)
  - Scrum
  - …

Waterfall Model



Characteristics of Waterfall Model

- Documentation-driven
- Strict sequential phases
- Well established
- Management perspective

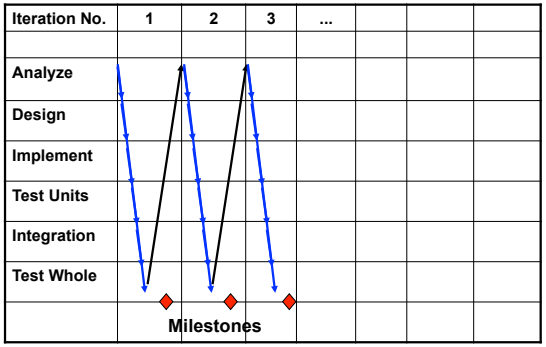
Pro's and Con's of Waterfall Model

- Advantages
  - Clear milestones
  - Well documented
  - Maintenance easier
  - Integrated testing
- Disadvantages
  - Large complex bureaucratic specifications
  - Inflexible against requirements change
  - Not realistic for larger projects
  - No risk handling
  - Bad user involvement
  - Separation of user and developer view
  - Product visible at end of project

Incremental & Iterative Models

- Address following issues:
  - Late requirements change
  - Tackle risks early
  - Early ROI
- Concepts
  - Develop product in small steps
  - Build product incrementally
  - Divide project into ***builds***

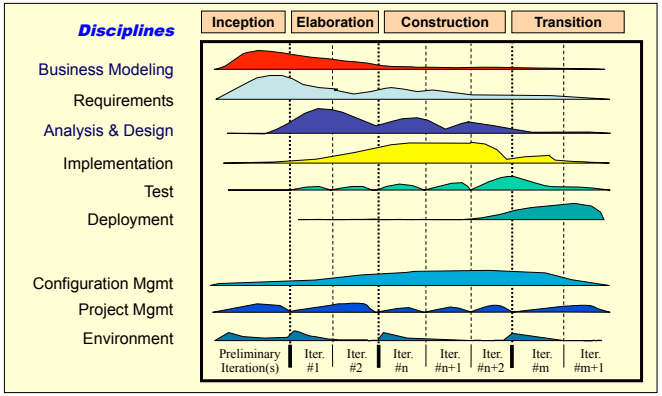
Principle of Incremental Model



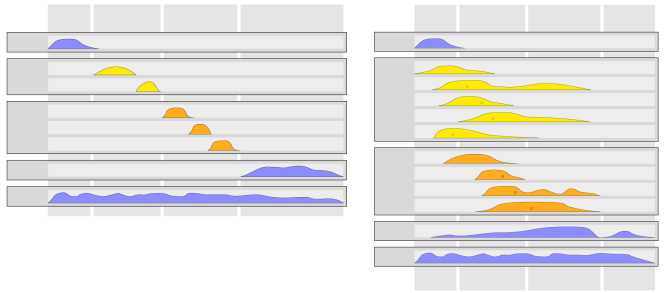
Pro's and Con's of I&I Model

- **Advantages**
  - Rapid operational quality portion of product
  - Less „traumatic“ for user
  - Rapid return on invest
  - Allows late requirement changes
  - Early risk detection
- **Disadvantages**
  - “*Build-And-Fix*” danger
  - Demanding project management
  - Management reservation

Example: Rational Unified Process (RUP)



Comparison: Sequential vs. Iterative



Zusamenfassung

Zusammenfassung

- Der Softwareentwicklungsprozess ist das Zusammenspiel von
  - Aktivitäten (Tätigkeiten)
  - Artefakten (Ergebnissen)
  - Rollen (Personen)
- Ein Vorgehensmodell beschreibt, wie diese Komponenten
  - in Beziehung stehen
  - in welcher Reihenfolge sie durchgeführt werden
  - wie sie geplant werden

Zusammenfassung

- Es gibt zwei Typen von Vorgehensmodellen
  - Sequenziell (sequential, predictive)
  - Iterativ, inkrementell (adaptive)

Zusammenfassung

- Sequenziell (z.B. Wasserfall)
  - Geht davon aus, dass Anforderungen präzise und abschliessend spezifiziert werden können
  - Das Produkt ist für die Anwender erst zum Schluss sichtbar
  - Probleme werden erst spät erkannt
  - Anwender sind schlecht einbezogen
  - Funktioniert nur wirklich für kurze überschaubare Projekte
  - Wird trotzdem oft verwendet, da gut zu kontrollieren und verwalten

Zusammenfassung

- Iterativ, inkrementell (z.B. RUP)
  - Geht davon aus, dass Änderungen unausweichlich sind und schliesst die Behandlung dieser in die Planung ein
  - Das Produkt wird stückweise erstellt und kann nach jeder Iteration ausprobiert werden
  - Probleme werden früh erkannt
  - Kann man als Aneinanderreihung von Mini-Wasserfällen betrachten
  - Anwender sind gut einbezogen
  - Kontrolle und Verwaltung sind anspruchsvoller

Part I:

The Software Development Process

Part II:

Requirements

Part III:

Use Cases

Analysis vs. Design

- Analysis
  - Find and describe objects and concepts in the problem domain
  - „Doing the right things“
- Design
  - Define software objects and how they collaborate to fulfill the requirements
  - „Doing things right“

Analysis – „Doing the right things“

- Artifacts
  - Vision
  - Use cases (user analysis, task analysis)
  - Supplementary specification
  - Domain model
  - Glossary
  - Prototypes, proof-of-concepts
  - Risk list
- Software Requirements Specification (SRS) document

What is a requirement?

- Condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents

or more user-centered:

- Condition or capability needed by a user to solve a problem or achieve an objective

Where do they come from?

- Requirements come from users and other stakeholders
- Stakeholders have demands
- Analyst must
  - Elicit the demands
  - Analyze them for consistency, feasibility, and completeness
  - Formulate them as requirements

What is a good requirement?

- unambiguous
- complete
- consistent
- necessary
- verifiable

What are the requirements for an alarm clock?

- Er muss genau getaktet sein
- Gehäuse nicht zu gross, handlich
- Robust
- Sollte guten Stand haben
- Muss Zeit auf Minute genau anzeigen

- Man muss Alarm einstellen können
- Man muss die Lautstärke des Alarms einstellen können
- Man muss Alarm abstellen können
- Sollte am Wochenende nicht läuten
- Man soll die Zeit einstellen können
- Sollte funkgesteuert sein
- Die Zeiger sollten leuchten

Different kinds or requirements: FURPS+

- **Functional**: what should the system do?
- **Non-functional**: how well should the system do it?
  - Usability
  - Reliability
  - Performance
  - Supportability
  - +
    - Implementation
    - Interface
    - Operations
    - Legal

Requirements Engineering

- Requirements engineering:
  - A systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system

1. **Elicit** requirements from various individual sources;
2. **Describe** the needs of all users and insure that they are consistent and feasible; and
3. **Validate** that the requirements so derived are an accurate reflection of user needs.

Requirements Elicitation Activities

1. **Identify** the relevant parties/actors (stakeholders) that are sources of requirements (end users, interfacing system, or environmental factors).
2. **Gather** the "wish list" for each relevant party (likely to be ambiguous, inconsistent, incomplete), analyze tasks, determine goals
3. **Document** and refine the "wish list" for each relevant party (repeatedly analyze until self-consistent).
4. **Integrate** the wish lists across the various relevant parties (check for feasibility, etc.)
5. **Determine** the nonfunctional requirements

Requirements Elicitation Techniques

- Techniques
  - Interviews
    - structured or unstructured
  - Contextual observations
  - Questionnaires
  - Rapid Prototyping
  - Analysis of work flows and artifacts (forms, guidelines)
  - Scenarios

Problems with requirements

- **Scope**: Requirements may address too little or too much information
  - Boundary of system ill defined
- **Understanding**: Within groups as well as between groups such as users and developers
  - Diverse stakeholders
- **Volatility**: i.e., the changing nature of requirements
  - User's needs evolve over time

How to Describe Requirements

- Feature List (traditional)
  - "System should do…"
  - Validation and verification straightforward
  - No connection to business goals
- Use Case Model
  - "User has goal … that product shall support."
  - In user's language, can reflect complexity
  - Design is harder (translation into features)
  - Doesn't cover non-task activities

Requirements Description

- Software Requirements Specification (SRS)
  - Describes the requirements in terms of functional and non-functional requirements
- Documentation Requirements
  - Traceability
    - Number **each** requirement
  - Ranking
    - Give **priority** to each requirement
  - Others
    - Consistency, completeness, non-ambiguity, ...

Zusamenfassung

Zusammenfassung

- Es gibt zwei verschiedene Arten von Anforderungen
  - Funktional: was soll das System tun können?
  - Nicht-funktional: wie gut soll das System etwas tun können?
- Nicht-funktionale Anforderungen (auch Qualitäts-, zusätzliche Anforderungen)
  - Benutzbarkeit
  - Zuverlässigkeit
  - Leistung
  - Implementierung, Betrieb, Gesetze, Schnittstellen

Zusammenfassung

- Es gibt zwei Arten von Beschreibungen für funktionale Anforderungen
  - Eigenschaftsliste (feature list)
    - "Das System soll … können"
      - Bsp.: "Die Ziffern sollen hell leuchten"
    - Keinen Bezug zu den Handlungen der Benutzenden
  - Anwendungsfall (use case)
    - "Der Benutzer hat das Ziel … welches vom System unterstützt werden soll"
      - Bsp.: "Die Ben. wollen die Zeit im Dunkeln erfahren können"
    - In der Sprache der Benutzenden; kann Komplexität erfassen
    - Kann Aktivitäten welche nicht durch Abläufe beschrieben werden können nicht abbilden (z.B. "Ben. wollen eine Übersicht erhalten")

Part I:

The Software Development Process

Part II:

Requirements

Part III:

Use Cases

What is a use case?

- Use cases describe how actors interact with the system to achieve a goal.

A Brief Use Case Example

- A customer arrives at the bank and asks the teller to deposit a certain amount of money on his account. The teller submits the necessary information to the system. The system performs the transaction and returns status information.

A More Formal Use Case Example

Use Case: Deposit Money
Goal: The intention of the Client is to deposit money on an account via a Teller. Many Clients may be performing transactions and queries at any one time.
Actors: Client, Teller
Main Scenario:
1. Client requests Teller to deposit money on an account, providing sum of money.
2. Teller requests System to perform a deposit, providing deposit transaction details*.
3. System validates the deposit, credits account for the amount, records details of the transaction, and informs Teller.

A "fully dressed" Use Case Example

Use Case: Deposit Money
Goal: The intention of the Client is to deposit money on an account via a Teller. Many Clients may be performing transactions and queries at any one time.
Actors: Client, Teller
Pre-Condition: Client has an account at the bank
Post-Condition: Money has been added to the account.
Main Scenario:
1. Client requests Teller to deposit money on an account, providing sum of money.
2. Teller requests System to perform a deposit, providing deposit transaction details*.
3. System validates the deposit, credits account for the amount, records details of the transaction, and informs Teller.

A "fully dressed" Use Case Example (contd)

Extensions:
2a. Client requests Teller to cancel deposit: use case ends in failure.
3a. System ascertains that it was given incorrect information:
    3a.1. System informs Teller; use case continues at step 2.
3b. System ascertains that it was given insufficient information to perform deposit:
    3b.1. System informs Teller; use case continues at step 2.
3c. System is not capable of depositing (e.g. transaction monitor of System is down)**:
    3c.1. System informs Teller; use case ends in failure.
Notes:
* a hyperlink to a document that contains data details and formats.
** this is an example of an IT infrastructure failure, we only write it in a use case if there is a corresponding project constraint that states a physical separation, e.g., transaction section depends on a legacy system which is located somewhere else.

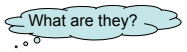
Use Case Description

- Use cases are primarily textual descriptions.
  - More than just an ellipse drawn in a UML diagram!
- Use case steps are written in an easy-to-understand structured narrative using the vocabulary of the application domain.
- Use cases are clear, precise, generalized, and technology-free descriptions.
- A use case sums up a set of scenarios:
  - Each scenario goes from trigger to completion.

Use Case Description

- It includes
  - How the use case starts and ends
  - The context of the use case
  - The actors and system behavior described as intentions and responsibilities
  - All the circumstances in which the primary actor's goal is reached and not reached
  - What information is exchanged

Actors



- An actor represents a role that an external entity such as a user, a hardware device, or another system plays in interacting with the system.
- A use case is not limited to a single actor.

Actors Categories

- Primary Actor:
  - actor with goal on system
  - obtains value from the system
- Secondary Actor:
  - actor with which the system has a goal
  - supports "creating value" for other actors

Use Cases

- Use cases offer a "familiar" representation to stakeholders
  - informal, easy to use, and story-telling-like style encourages them to be actively involved in defining the requirements;
  - thus, easier to validate with stakeholders;
  - allows common understanding between developers, system end users, and domain experts—"Is this what you want?".
- They are scalable:
  - Use cases can be decomposed/composed—each step is *ideally* a sub-goal.

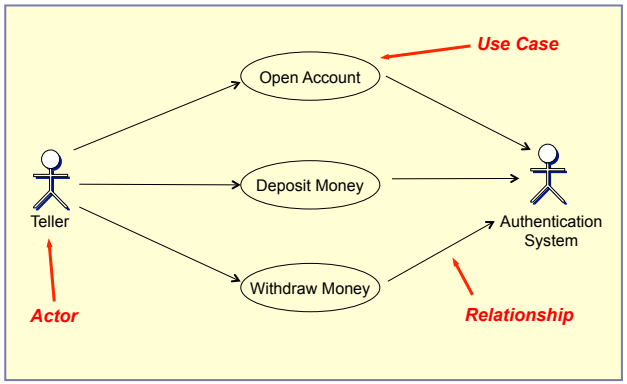
Use Cases

- Being a black-box view of the system, use cases are a good approach for finding the *What* rather than the *How*.
  - A black-box matches users view of the system: things going in and things coming out.
- Use cases force one to look at exceptional as well as normal behavior.
  - helps us to surface hidden requirements
- Use cases can help formulate system tests.
  - "Is this use case built into the system?"

Use Cases

- Replace the monotonous requirements list
  - use cases define all functional requirements
  - easier (and more intrinsically interesting) to extract user goals than list a bunch of "shall" statements
- Use case templates facilitate interviewing and reviews
- Ease an iterative development lifecycle
  - levels of precision for a use case by refinement
- Support an incremental development lifecycle
  - E.g. "Acme" Release 1: use cases 1-20;
    "Acme" Release 2: use cases 1-29.

A UML Use Case Diagram



Zusamenfassung

Zusammenfassung

- Ein Use Case (deutsch: Anwendungsfall) beschreibt wie ein Akteur mit dem System zusammenarbeitet um ein Ziel zu erreichen.
- Ein Use Case beschreibt wer (Akteur) mit dem System was macht (Interaktion) und mit welcher Absicht (Ziel), ohne sich um interne Systemdetails zu kümmern.
- Ein Use Case ist in der Sprache der Problemstellung geschrieben und enthält keine Implementationsdetails.

Zusammenfassung

- Ein Akteur repräsentiert eine Rolle die eine externe Einheit (Benutzer, Gerät, anderes System) spielt während dem sie mit dem System interagiert.
- Ein Use Case kann mehr als einen Akteur haben
- Zwei Kategorien:
  - Primäre Akteure
    - Haben eine konkrete Absicht mit dem System
    - Haben einen Nutzen von der Interaktion mit dem System
  - Sekundäre Akteure, unterstützende Akteure
    - Akteure mit welchen das System eine Absicht hat
    - Unterstützen die Nutzenerbringung für die primären Akteure

Zusammenfassung

- Eine Use Case Beschreibung besteht aus:
  - Dem Kontext des Use Cases (Ziel, Interessen der Anspruchsgruppen)
  - Den beteiligten Akteuren
  - Wie er beginnt und endet
  - Dem Verhalten der Akteure und des Systems, formuliert als Absichten und Verantwortlichkeiten
  - Allen Umständen unter denen der primäre Akteur sein Ziel ereicht oder nicht erreicht
  - Welcher Information ausgetauscht wird

Zusammenfassung

- Ein Use Case Diagramm wird dazu verwendet um einen Überblick über alle Use Cases und deren Akteure zu geben. Es ersetzt nicht die strukturierten textuellen Beschreibungen der einzelnen Use Cases.