Requirements Engineering

Requirement dan Isinya

Topik

- Definisi Requirement
- Requirement Fungsional dan nonfungsional
- Dokumen Spesifikasi Requirement Software (SRS)
- Proses Rekayasa Requirement
- Analisa dan eksplorasi requirement
- Validasi Requirement
- Manajemen Requirement

Requirements engineering

- Proses untuk menganalisa dan merangkum layanan yang dituntut oleh customer terhadap sistem dan batasan di mana sistem tersebut dikembangkan dan beroperasi.
- Requirements sendiri adalah deskripsi dari layanan dan batasan dari sistem yang dihasilkan dari proses requirement engineering.
- Fase requirement engineering diakhiri dengan hasil berupa dokumen Software requirement Specification (SRS)

Apakah Requirement itu?

- Definisi IEEE (Institute of Electrical and Electronics Engineers) : kondisi atau kemampuan yang harus dimiliki oleh sistem
- Bisa meliputi dari abstraksi high-level dari layanan, batasan sistem, hingga ke spesifikasi fungsi matematika secara detil.
- Mutlak diperlukan, requirements bisa berperan sebagai:
  - Sebagai basis untuk pengajuan penawaran terhadap lelang kontrak: - harus open untuk interpretasi;
  - Merupakan isi kontrak itu sendiri: harus detil;

Requirements abstraction (Davis,1993)

“If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization’s needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system.”

Arti Penting SRS (Software Requirement Specification)

- SRS menjadi basis kesepakatan antara user dan pembuat software.
  - User memiliki kebutuhan yang harus dipenuhi, tapi mungkin tidak tahu software
  - Developers will develop the system, but may not know about problem domain
  - SRS is the medium to bridge the commn. gap and specify user needs in a manner both can understand

Arti Penting SRS (Software Requirement Specification)

- Membantu user mengetahui kebutuhannya.
  - User tidak selalu tahu kebutuhannya
  - Harus dianalisa dan dipahami potensialnya
  - Tujuan hendaknya tidak hanya sekedar mengotomatisasi sistem manual, tapi memberi nilai tambah melalui IT
  - Proses requirement membantu memperjelas kebutuhan
- SRS merupakan referensi untuk validasi produk akhir
  - Kejelasan apa yang diharapkan.
  - Validasi: “ SW memenuhi SRS “

Arti Penting SRS (Software Requirement Specification)

- High quality SRS essential for high Quality SW
  - Requirement errors get manifested in final sw
  - to satisfy the quality objective, must begin with high quality SRS
  - Requirements defects are not few
    - 25% of all defects in one case; 54% of all defects found after UT
    - 80 defects in A7 that resulted in change requests
    - 500 / 250 defects in previously approved SRS.

Arti Penting SRS (Software Requirement Specification)

- SRS yang bagus mengurangi biaya pengembangan
  - Kesalahan pada SRS akan mahal jika diperbaiki kemudian
  - Perubahan requirement bisa berakibat biaya meningkat tajam (up to 40%)
  - SRS yang bagus bisa meminimalisir perubahan dan error
  - Ekstra tenaga yang digunakan untuk penghematan pada fase requirement akan menghasilkan penghematan yang berlipat dibandingkan tenaga yang dikeluarkan
- Sebagai contoh
  - Biaya untuk memperbaiki error secara berturut-turut dari requirement, desain, koding, test penerimaan, dan pengoperasian adalah 2 , 5 , 15 , 50 , 150 person-hours

Arti Penting SRS (Software Requirement Specification)

- Contoh …
  - Setelah fase requirement 65 requirement error ditemukan pada waktu desain, 2 pada waktu koding, 30 ketika testing penerimaan, dan 3 selama pengoperasian
  - Jika 50% requirement errors tidak dihilangkan sejak di fase requirement, total biayanya adalah 32.5 *5 + 1*15 + 15*50 + 1.5*150 = 1152 jam
  - Jika tenaga ekstra 100 person-hours diinvestasikan pada fase req untuk menemukan 50% error ini, maka biaya pengembangan bisa dikurangi 1152 person-hours.
  - Net biaya bisa dikurangi 1052 person-hours

Tipe Requirement

- User requirements
  - Pernyataan dalam bahasa natural ditambah diagram mengenai layanan yang disediakan oleh sistem dan batasan pengoperasiannya. Ditulis oleh customer.

- System requirements
  - Dokumen terstruktur yang menjelaskan secara detil fungsi dari sistem, layanan, dan batasan operasinya. Harus berisi penjelasan mana yang akan diimplementasikan. Bisa masuk dalam kontrak antara klien dan pembuat software.

- User requirement sifatnya lebih umum

User and system requirements

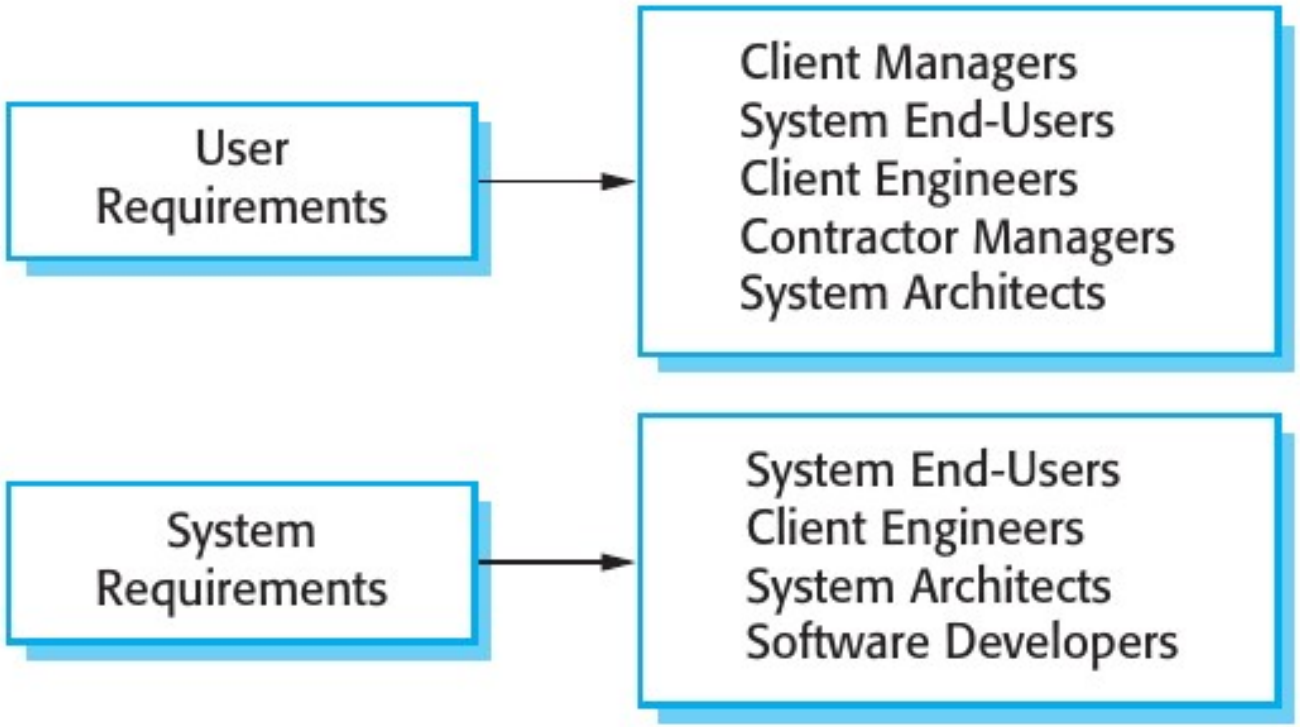
User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Readers of different types of requirements specification



Functional dan non-functional requirements

- Functional requirements
  - Pernyataan tentang layanan yang harus disediakan oleh sistem, bagaimana sistem merespon input tertentu, dan bagaimana perilaku sistem ketika menghadapi situasi tertentu.
  - Bisa juga secara eksplisit menegaskan apa yang tidak dilakukan oleh sistem.
- Non-functional requirements
  - Batasan dari layanan yang disediakan oleh sistem, seperti waktu, batasan proses pengembangan, standard, dll.
  - Seringnya berlaku untuk sistem sebagai keseluruhan, bukan pada fitur atau layanan individual dari sistem.
- Domain requirements
  - Constraints on the system from the domain of operation

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Example functional requirements for the Software Medical Health Care-Patient Management System

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘search’ in requirement 1
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

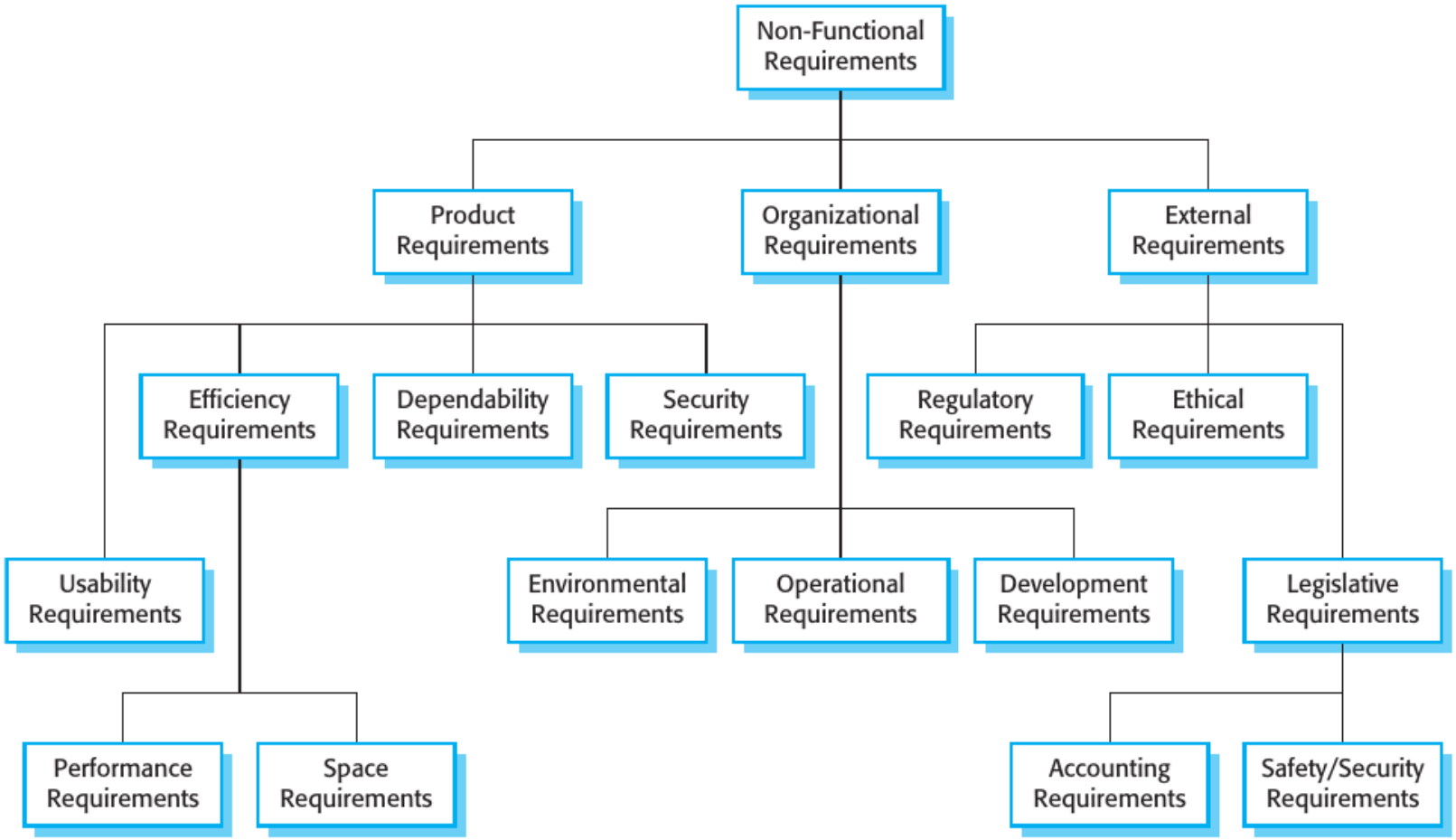
Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
- Complete
  - They should include descriptions of all facilities required.
- Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, it is impossible to produce a complete and consistent requirements document.

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of nonfunctional requirement



Non-functional requirements implementation

- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements.

Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Examples of nonfunctional requirements in the MHC-PMS

Product requirement
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

External requirement
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Goals and requirements

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal
  - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.

Usability requirements

- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)
- Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Domain requirements

- The system's operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Train protection system

- This is a domain requirement for a train protection system:
- The deceleration of the train shall be computed as:
  - Dtrain = Dcontrol + Dgradient
  - where Dgradient is 9.81ms2 * compensated gradient/alpha and where the values of 9.81ms2 /alpha are known for different types of train.
- It is difficult for a non-specialist to understand the implications of this and how it interacts with other requirements.

Domain requirements problems

- Understandability
  - Requirements are expressed in the language of the application domain;
  - This is often not understood by software engineers developing the system.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

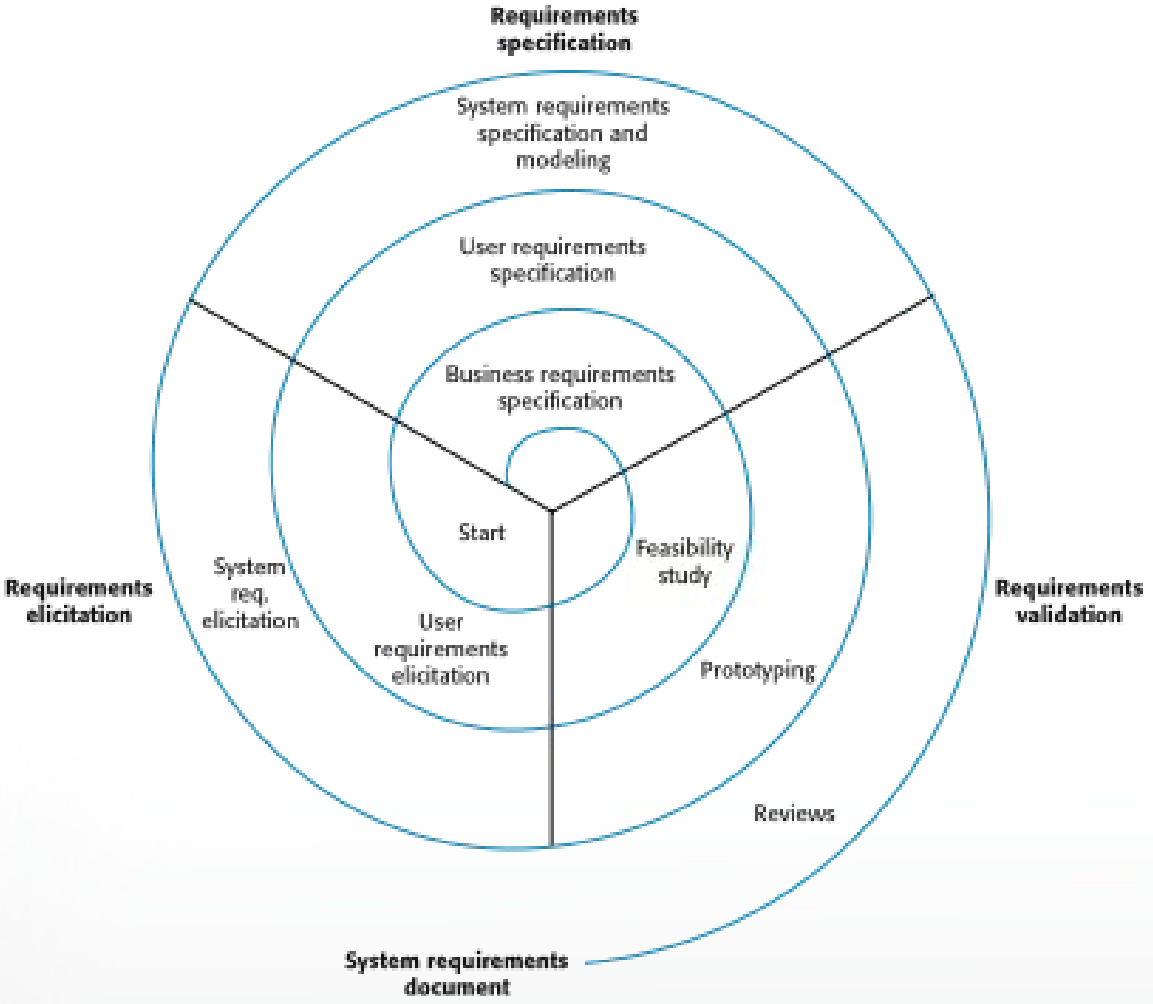
Key points

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Requirements Engineering

Analisa dan Eksplorasi Requirement

A spiral view of the requirements engineering process



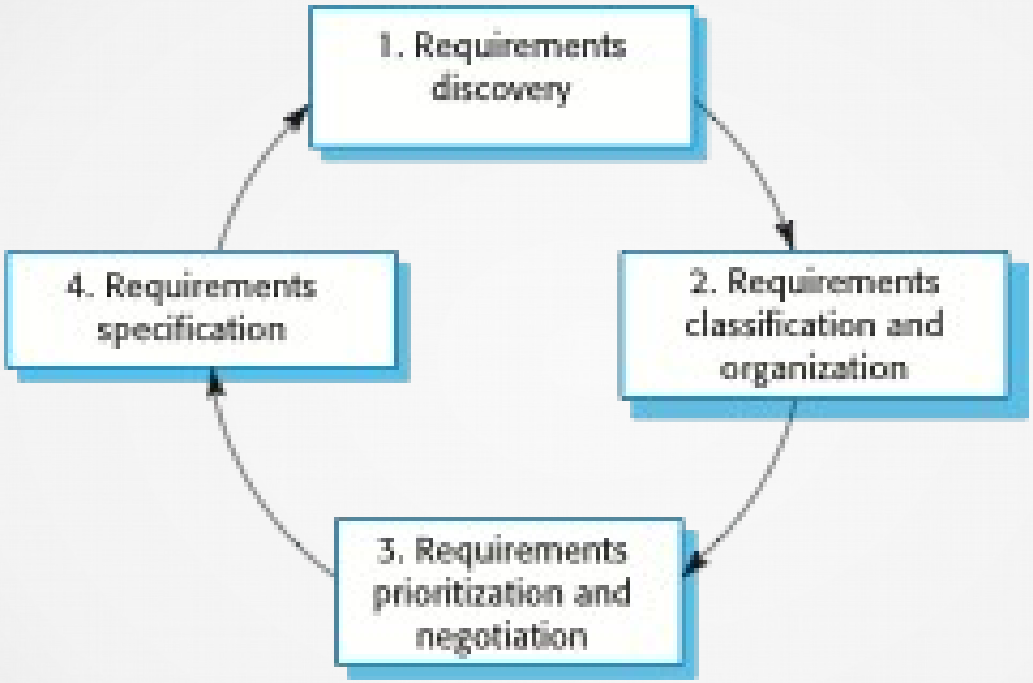
Requirements elicitation and analysis (analisa dan eksplorasi requirement)

- Melibatkan staf teknis, bekerja dengan customer untuk mengetahui domain aplikasi, layanan/fungsi dari sistem, dan batasan operasional
- Juga bisa melibatkan pegguna akhir (operator), manajer, insinyur yang menangani pemeliharaan, tenaga ahli pada domain sistem, organisasi terkait, dll. Mereka ini disebut *stakeholders.*

Problem yang bisa muncul

- Stakeholders tidak mengerti apa yang benar-benar mereka inginkan.
- Stakeholders mengekspresikan requirement dengan cara mereka masing-masing.
- Stakeholders berbeda, mungkin mempunyai requirement yang saling bertentangan.
- Faktor organisasi dan politik bisa mempengaruhi requirement sistem.
- Perubahan requirement bisa terjadi ketika dalam proses analisa. Stakeholders bisa datang dan merubah kondisi bisnis.

The requirements elicitation and analysis process



Process activities

- Requirements discovery
  - Interaksi dengan stakeholders untuk eksplorasi requirement dari mereka. Domain requirement juga bisa tereksplor pada tahap ini.
- Requirements classification and organisation
  - Mengelompokkan requirement yang berkaitan dan mengorganisirnya dalam kluster yang logis dan mudah dimengerti
- Prioritisation and negotiation
  - Prioritasi requirement dan memecahkan konflik yang mungkin terjadi antar requirement.
- Requirements specification
  - Requirement didokumentasikan dalam software requirement specification dan menjadi input untuk spiral selanjutnya.

Software requirement Specification (SRS)

- Proses untuk menganalisa dan merangkum layanan yang dituntut oleh customer terhadap sistem dan batasan di mana sistem tersebut dikembangkan dan beroperasi diakhiri dengan hasil berupa dokumen Software requirement Specification (SRS)

Struktur dari SRS (Software Requirement Specification)

- Pengantar
  - Tujuan, tujuan dasar dari sistem
  - Scope (ruang lingkup) yang dikerjakan sistem, dan apa yang tidak dikerjakan oleh sistem
  - Gambaran umum, gambaran singkat tentang isi SRS
- Deskripsi keseluruhan
  - Perspektif produk
  - Fungsi produk
  - Karakteristik pengguna
  - Asumsi dan ketergantungan
  - Constraints

Struktur dari SRS…

- Requirement spesifik
  - Interface eksternal
  - Requirement fungsional
  - Requirement performa
  - Design constraints
- Kriteria akseptabel
  - Krtiteria tambahan yang hendak dispesifikasikan di muka.
- Standardisasi SRS ini dilakukan oleh IEEE.

Penggunaan Use Cases untuk spesifikasi Requirement Fungsional

- Traditional approach for fn specs – specify each function
- Use cases is a newer technique for specifying behavior (functionality)
- I.e. focuses on functional specs only
- Though primarily for specification, can be used in analysis and elicitation
- Can be used to specify business or org behavior also, though we will focus on sw
- Well suited for interactive systems

Use Cases Basics

- A use case captures a contract between a user and system about behavior
- Basically a textual form; diagrams are mostly to support
- Also useful in requirements elicitation as users like and understand the story telling form and react to it easily

Basics..

- Actor: a person or a system that interacts with the proposed system to achieve a goal
  - Eg. User of an ATM (goal: get money); data entry operator; (goal: Perform transaction)
- Actor is a logical entity, so receiver and sender actors are different (even if the same person)
- Actors can be people or systems
- Primary actor: The main actor who initiates a UC
  - UC is to satisfy his goals
  - The actual execution may be done by a system or another person on behalf of the Primary actor

Basics..

- Scenario: a set of actions performed to achieve a goal under some conditions
  - Actions specified as a sequence of steps
  - A step is a logically complete action performed either by the actor or the system
- Main success scenario – when things go normally and the goal is achieved
- Alternate scenarios: When things go wrong and goals cannot be achieved

Basics..

- A UC is a collection of many such scenarios
- A scenario may employ other use cases in a step
- I.e. a sub-goal of a UC goal may be performed by another UC
- I.e. UCs can be organized hierarchically

Basics…

- UCs specify functionality by describing interactions between actors and system
- Focuses on external behavior
- UCs are primarily textual
  - UC diagrams show UCs, actors, and dependencies
  - They provide an overview
- Story like description easy to understand by both users and analysts
- They do not form the complete SRS, only the functionality part

Example

Use Case 1: Buy stocks

Primary Actor: Purchaser

Goals of Stakeholders:

- Purchaser: wants to buy stocks
- Company: wants full transaction info

Precondition: User already has an account

Example …

- Main Success Scenario
  1. User selects to buy stocks
  2. System gets name of web site from user for trading
  3. Establishes connection
  4. User browses and buys stocks
  5. System intercepts responses from the site and updates user portfolio
  6. System shows user new portfolio stading

Example…

- Alternatives
  - 2a: System gives err msg, asks for new suggestion for site, gives option to cancel
  - 3a: Web failure. 1-Sys reports failure to user, backs up to previous step. 2-User exits or tries again
  - 4a: Computer crashes
  - 4b: web site does not ack purchase
  - 5a: web site does not return needed info

Example 2

- Use Case 2: Buy a product
- Primary actor: buyer/customer
- Goal: purchase some product
- Precondition: Customer is already logged in

Example 2…

- Main Scenario
  1. Customer browses and selects items
  2. Customer goes to checkout
  3. Customer fills shipping options
  4. System presents full pricing info
  5. Customer fills credit card info
  6. System authorizes purchase
  7. System confirms sale
  8. System sends confirming email

Example 2…

- Alternatives
  - 6a: Credit card authorization fails
    - Allows customer to reenter info
  - 3a: Regular customer
    - System displays last 4 digits of credit card no
    - Asks customer to OK it or change it
    - Moves to step 6

Example – An auction site

- ***Use Case1:*** Put an item for auction
- ***Primary Actor:*** Seller
- ***Precondition:*** Seller has logged in
- ***Main Success Scenario:***
  - Seller posts an item (its category, description, picture, etc.) for auction
  - System shows past prices of similar items to seller
  - System specifies the starting bid price and a date when auction will close
  - System accepts the item and posts it
- ***Exception Scenarios:***
  - -- 2 a) There are no past items of this category
    - * System tells the seller this situation

Example – auction site..

- ***Use Case2:*** Make a bid
- ***Primary Actor:*** Buyer
- ***Precondition:*** The buyer has logged in
- ***Main Success Scenario:***
  - Buyer searches or browses and selects some item
  - System shows the rating of the seller, the starting bid, the current bids, and the highest bid; asks buyer to make a bid
  - Buyer specifies bid price, max bid price, and increment
  - Systems accepts the bid; Blocks funds in bidders account
  - System updates the bid price of other bidders where needed, and updates the records for the item

- ***Exception Scenarios:***
  - -- 3 a) The bid price is lower than the current highest

    * System informs the bidder and asks to rebid

  - -- 4 a) The bidder does not have enough funds in his account

    * System cancels the bid, asks the user to get more funds

Example –auction site..

- ***Use Case3:* Complete auction of an item**
- ***Primary Actor:*** Auction System
- ***Precondition:*** The last date for bidding has been reached
- ***Main Success Scenario:***
  - Select highest bidder; send email to selected bidder and seller informing final bid price; send email to other bidders also
  - Debit bidder's account and credit seller's account
  - Transfer from seller's account commission amount to organization's account
  - Unblock other bidders funds
  - Remove item from the site; update records
- ***Exception Scenarios:*** None

Example – summary-level Use Case

- ***Use Case 0* : Auction an item**
- ***Primary Actor:*** Auction system
- ***Scope:*** Auction conducting organization
- ***Precondition:*** None
- ***Main Success Scenario:***
  - Seller performs put an item for auction
  - Various bidders make a bid
  - On final date perform Complete the auction of the item
  - Get feed back from seller; get feedback from buyer; update records

Requirements with Use Cases

- UCs specify functional requirements
- Other req identified separately
- A complete SRS will contain the use cases plus the other requirements
- Note – for system requirements it is important to identify UCs for which the system itself may be the actor

Developing Use Cases

- UCs form a good medium for brainstorming and discussions
- Hence can be used in elicitation and problem analysis also
- UCs can be developed in a stepwise refinement manner
  - Many levels possible, but four naturally emerge

Developing…

- Step 1: Identify actors and goals
  - Prepare an actor-goal list
  - Provide a brief overview of the UC
  - This defines the scope of the system
  - Completeness can also be evaluated
- Step 2: Specify main Success Scenarios
  - For each UC, expand main scenario
  - This will provide the normal behavior of the system
  - Can be reviewed to ensure that interests of all stakeholders and actors is met

Developing…

- Step 3: Identify failure conditions
  - List possible failure conditions for UCs
  - For each step, identify how it may fail
  - This step uncovers special situations
- Step 4: Specify failure handling
  - Perhaps the hardest part
  - Specify system behavior for the failure conditions
  - New business rules and actors may emerge

Pendekatan analisis selain Use Case

Data Flow Modeling

- Widely used; focuses on functions performed in the system
- Views a system as a network of data transforms through which the data flows
- Uses data flow diagrams (DFDs) and functional decomposition in modeling
- The SSAD methodology uses DFD to organize information, and guide analysis

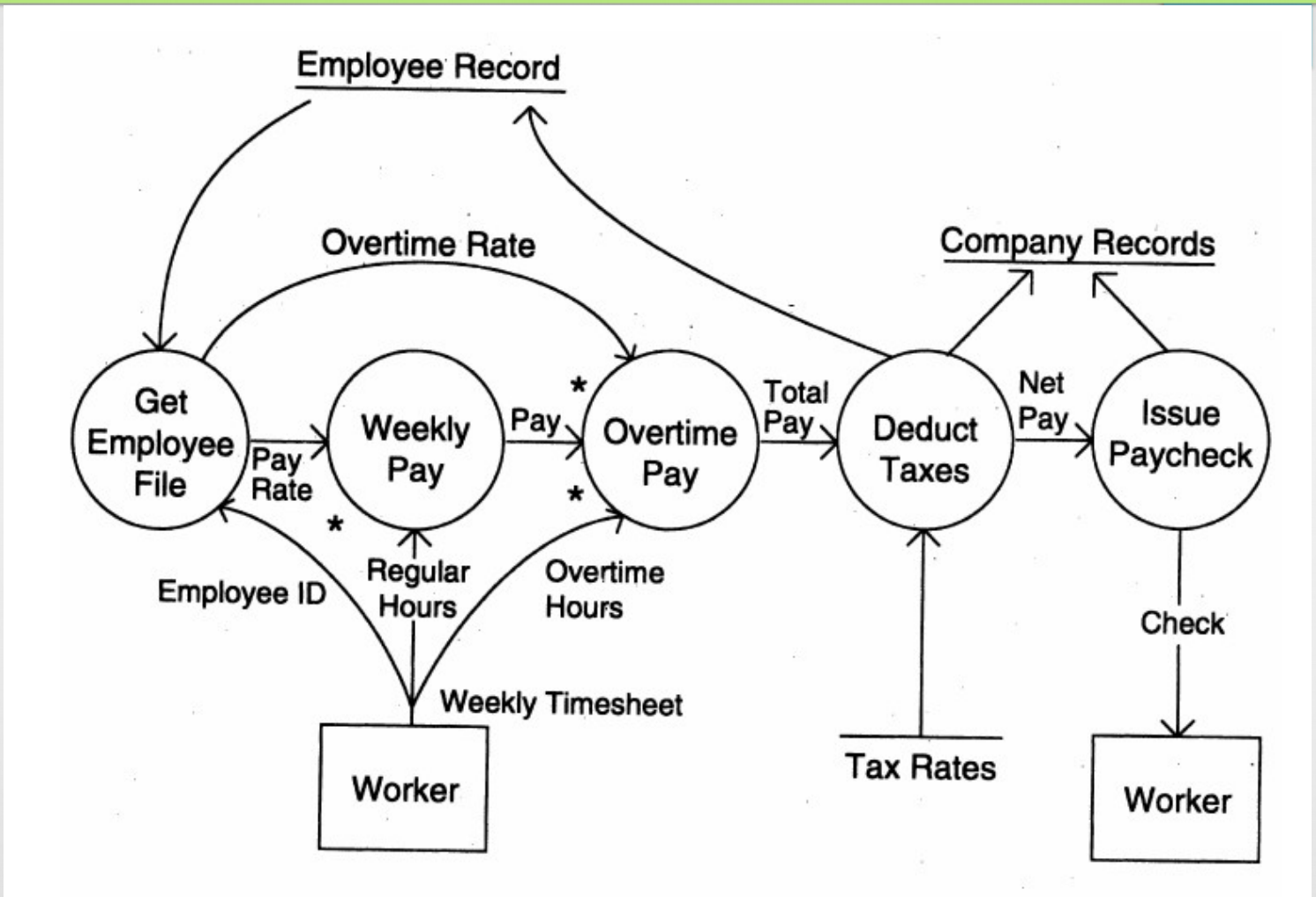
Data flow diagrams

- A DFD shows flow of data through the system
  - Views system as transforming inputs to outputs
  - Transformation done through transforms
  - DFD captures how transformation occurs from input to output as data moves through the transforms
  - Not limited to software

Data flow diagrams…

- DFD
  - Transforms represented by named circles/bubbles
  - Bubbles connected by arrows on which named data travels
  - A rectangle represents a source or sink and is originator/consumer of data (often outside the system)

DFD Example


Employee Record
Overtime Rate
Company Records
Get Employee File
Pay Rate
Weekly Pay
Pay
Overtime Pay
Total Pay
Deduct Taxes
Net Pay
Issue Paycheck
*
*
*
Employee ID
Regular Hours
Overtime Hours
Weekly Timesheet
Check
Worker
Tax Rates
Worker

DFD Conventions

- External files shown as labeled straight lines
- Need for multiple data flows by a process represented by * (means and)
- OR relationship represented by +
- All processes and arrows should be named
- Processes should represent transforms, arrows should represent some data

Data flow diagrams…

- Focus on what transforms happen , how they are done is not important
- Usually major inputs/outputs shown, minor are ignored in this modeling
- No loops , conditional thinking , …
- DFD is NOT a control chart, no algorithmic design/thinking
- Sink/Source , external files

Drawing a DFD

- If get stuck , reverse direction

- If control logic comes in , stop and restart
- Label each arrows and bubbles
- Make use of + & *
- Try drawing alternate DFDs

<u>Leveled DFDs :</u>

- DFD of a system may be very large
- Can organize it hierarchically
- Start with a top level DFD with a few bubbles
- then draw DFD for each bubble
- Preserve I/O when “ exploding”

Drawing a DFD for a system

- Identify inputs, outputs, sources, sinks for the system
- Work your way consistently from inputs to outputs, and identify a few high-level transforms to capture full transformation
- If get stuck, reverse direction
- When high-level transforms defined, then refine each transform with more detailed transformations

Drawing a DFD for a system..

- Never show control logic; if thinking in terms of loops/decisions, stop & restart
- Label each arrows and bubbles; carefully identify inputs and outputs of each transform
- Make use of + & *
- Try drawing alternate DFDs

Leveled DFDs

- DFD of a system may be very large
- Can organize it hierarchically
- Start with a top level DFD with a few bubbles
- then draw DFD for each bubble
- Preserve I/O when “ exploding” a bubble so consistency preserved
- Makes drawing the leveled DFD a top-down refinement process, and allows modeling of large and complex systems

Data Dictionary

- In a DFD arrows are labeled with data items
- Data dictionary defines data flows in a DFD
- Shows structure of data; structure becomes more visible when exploding

- Can use regular expressions to express the structure of data

Data Dictionary Example

- For the timesheet DFD

Weekly_timesheet – employee_name + id + [regular_hrs + overtime_hrs]*

Pay_rate = [hourly | daily | weekly] + dollar_amt

Employee_name = last + first + middle

Id = digit + digit + digit + digit

DFD drawing – common errors

- Unlabeled data flows
- Missing data flows
- Extraneous data flows
  - Consistency not maintained during refinement
  - Missing processes
  - Too detailed or too abstract
  - Contains some control information

Prototyping

- Prototyping is another approach for problem analysis
- Discussed it earlier with process – leads to prototyping process model

Requirements Validation

- Lot of room for misunderstanding
- Errors possible
- Expensive to fix req defects later
- Must try to remove most errors in SRS
- Most common errors
  - Omission - 30%
  - Inconsistency - 10-30%
  - Incorrect fact - 10-30%
  - Ambiguity - 5 -20%

Requirements Review

- SRS reviewed by a group of people
- Group: author, client, user, dev team rep.
- Must include client and a user
- Process – standard inspection process
- Effectiveness - can catch 40-80% of req. errors

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

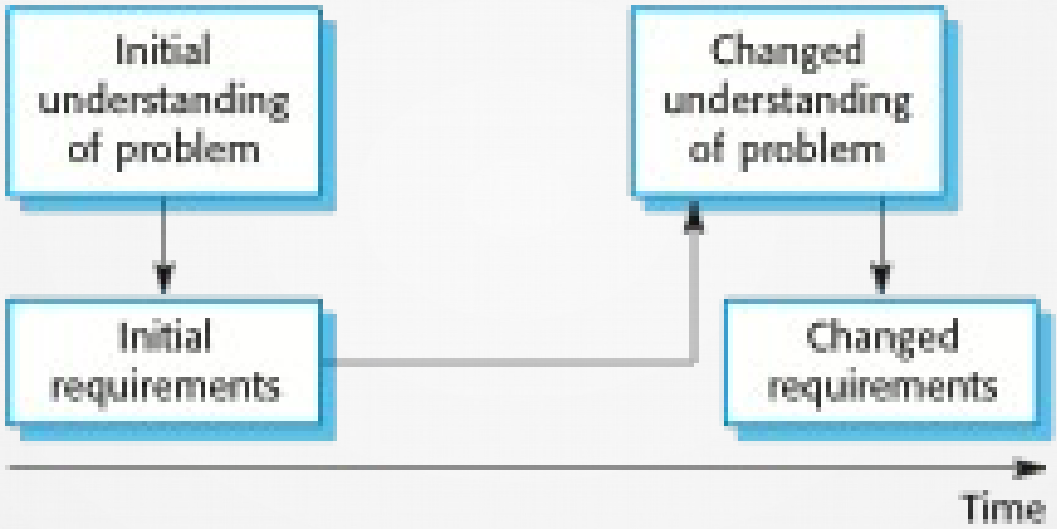
Changing requirements

- The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
  - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

Requirements evolution



Requirements management planning

- Establishes the level of requirements management detail that is required.
- Requirements management decisions:
  - *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
  - *A change management process* This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
  - *Traceability policies* These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
  - *Tool support* Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements change management

- Deciding if a requirements change should be accepted
  - *Problem analysis and change specification*
    - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
  - *Change analysis and costing*
    - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
  - Change implementation
    - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

Requirements change management