Software Engineering

Requirements Engineering

何明昕 HE Mingxin, Max

Send your email to c.max@yeah.net with
a subject like: SE-*id-Andy: On What …*

Download from c.program@yeah.net

/文件中心/网盘/SoftwareEngineering2024

Topics

- The Essential Software Requirement
- Requirements Development (majors)
- Requirements Management (minors)
- Main Requirements Engineering Components
- Requirements and User Stories
- Types of Requirements
- Effort Estimation (Agile Methods)

Requirements & Implementation

Requirements & Implementation

Requirements & Implementation

Requirements & Implementation

What are Requirements?

Our favorite definition
from Ian Sommerville and Pete Sawyer (1997):

Requirements are a specification of what should be implemented.

They are descriptions of how the system should behave,
or of a system property or attribute.

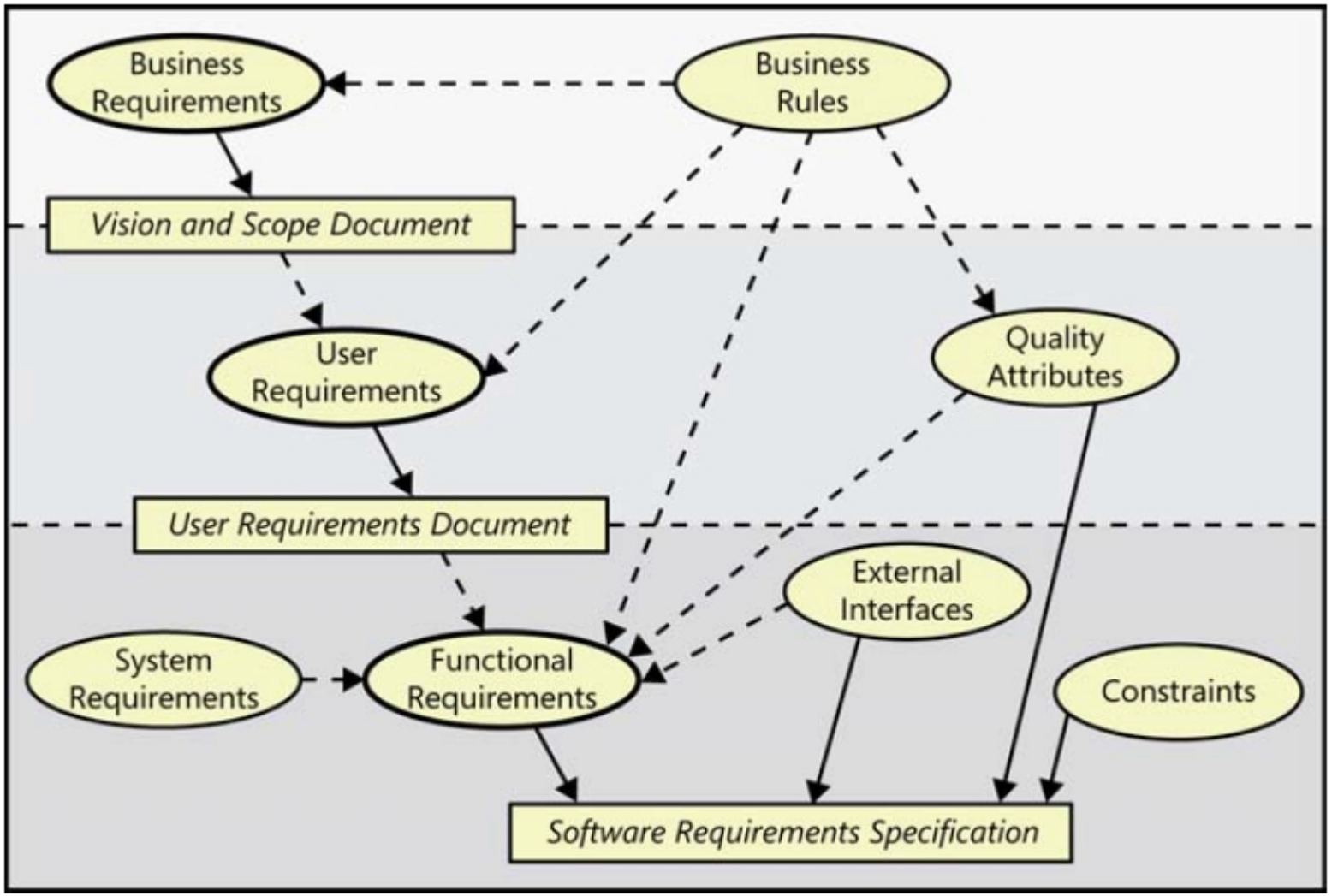
They may be a constraint on the development process of the system.

Software Requirements

- A **requirement** specifies the business functions that the user will be able to perform using the system-to-be in different "situations" or "contexts", and the kind of experience the user will have during this work
  - Other concerns, such as how the system will manage the resources (computing, network, …), how the system will manage and protect user's data, etc.
- User requirements will often be high-level, vague and incomplete. They are more like high-level goals, or business goals, rather than software requirements needed by the developer
- When trying to achieve a given high-level goal, we will need to consider what matters, what are the important parameters, so that we can derive the detailed technical requirements
- Only based on deeper understanding of detailed issues, we can identify important "scenarios" or "situations" and identify what parameters should be considered in each situation
- Then using these parameters, we decide what the system should do, or how to respond to this situation (i.e., inputs)

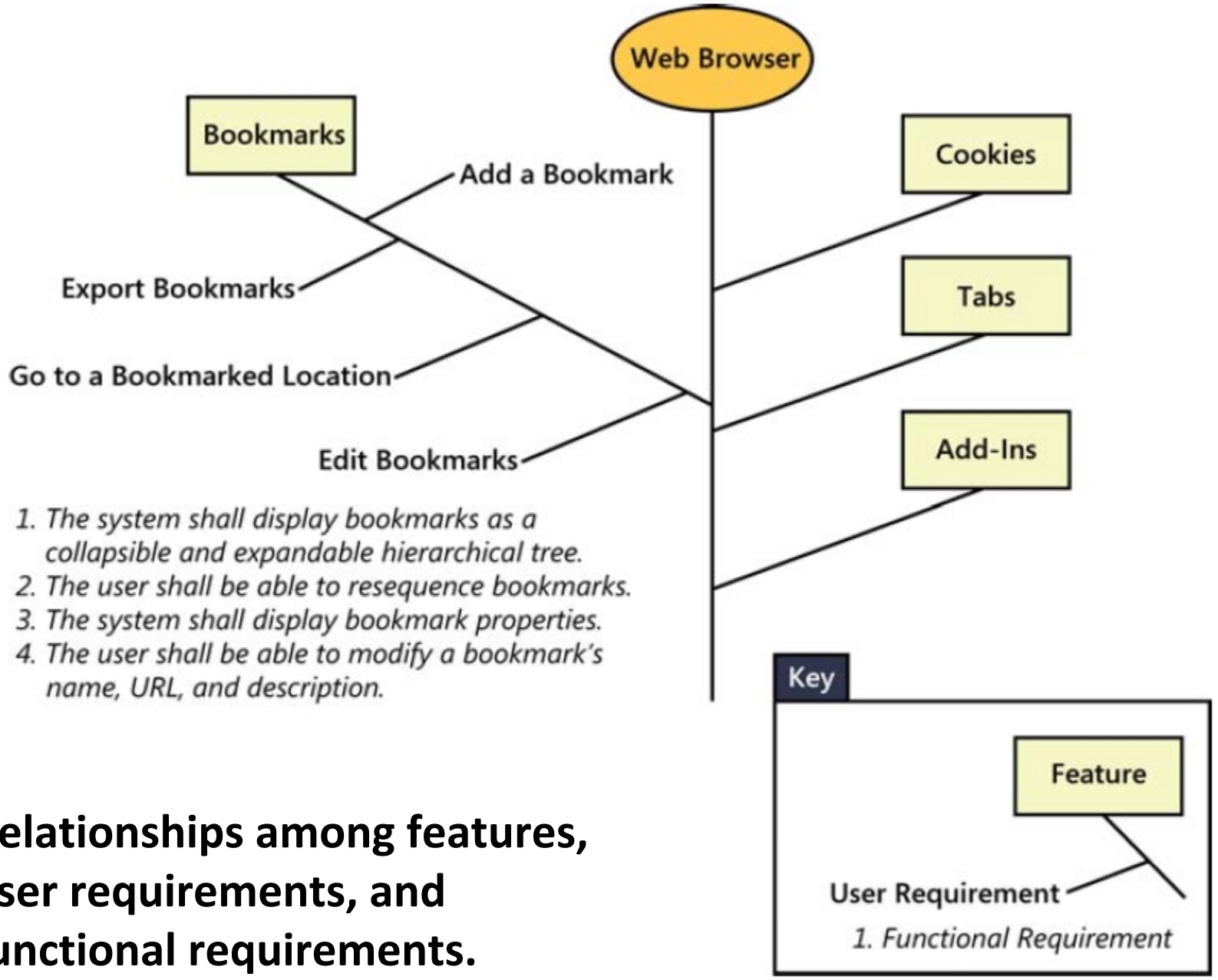
Some types of requirements information

Term	Definition
Business requirement	A high-level business objective of the organization that builds a product or of a customer who procures it.
Business rule	A policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself, but the origin of several types of software requirements.
Constraint	A restriction that is imposed on the choices available to the developer for the design and construction of a product.
External interface requirement	A description of a connection between a software system and a user, another software system, or a hardware device.
Feature	One or more logically related system capabilities that provide value to a user and are described by a set of functional requirements.
Functional requirement	A description of a behavior that a system will exhibit under specific conditions.
Nonfunctional requirement	A description of a property or characteristic that a system must exhibit or a constraint that it must respect.
Quality attribute	A kind of nonfunctional requirement that describes a service or performance characteristic of a product.
System requirement	A top-level requirement for a product that contains multiple subsystems, which could be all software or software and hardware.
User requirement	A goal or task that specific classes of users must be able to perform with a system, or a desired product attribute.



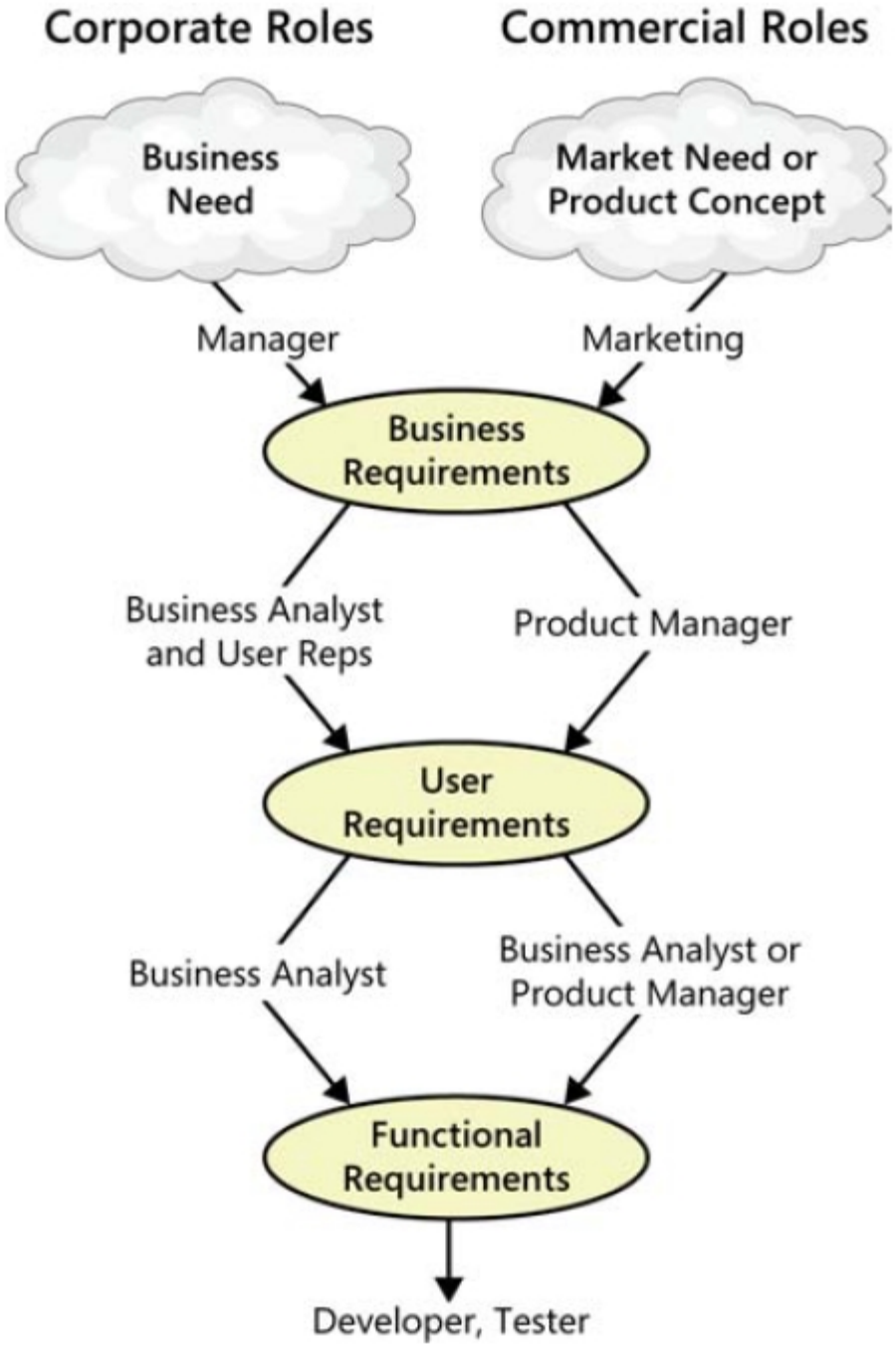
Relationships among several types of requirements information.

Solid arrows mean "are stored in";
dotted arrows mean "are the origin of" or "influence."



Relationships among features, user requirements, and functional requirements.

An example of how different stakeholders participate in requirements development.

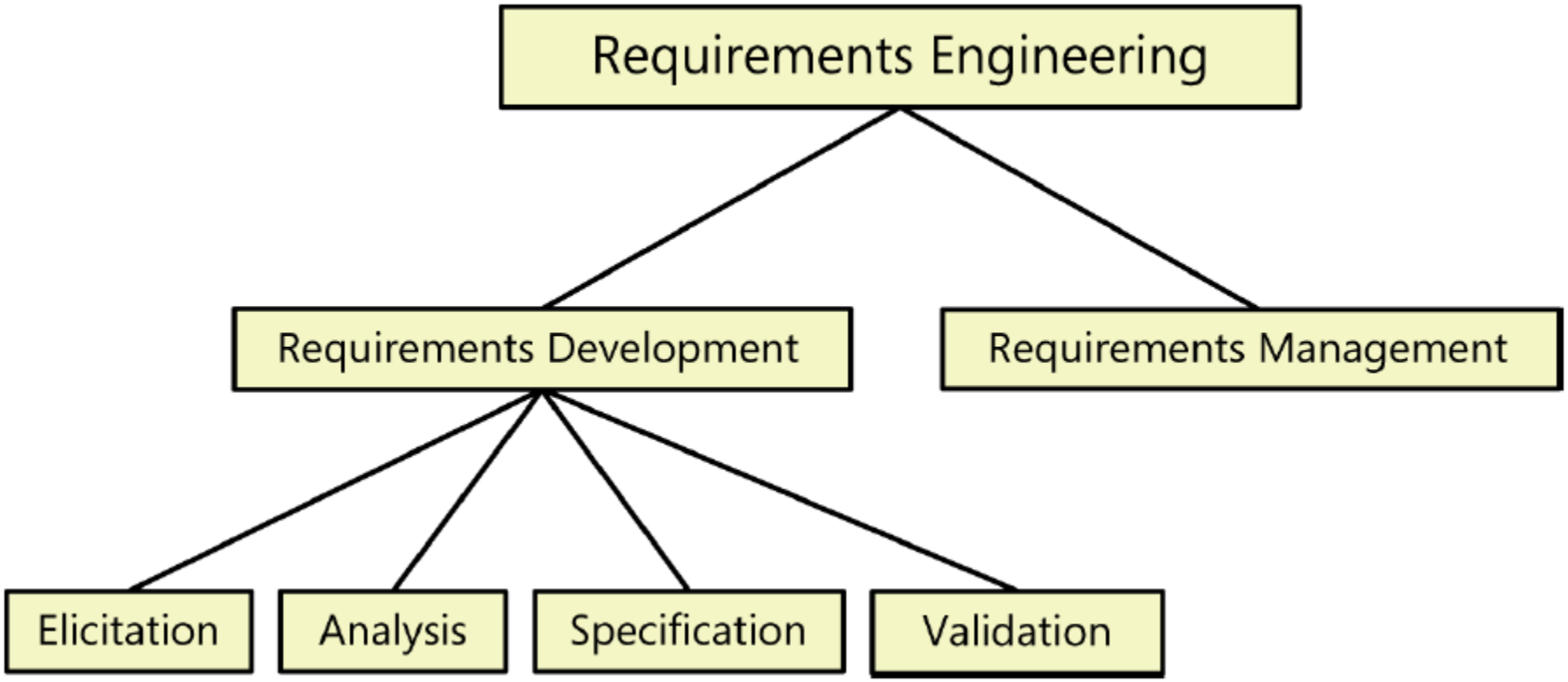


Product (majors) vs. project (minors) requirements

Project requirements include:

- Physical resources the development team needs, such as workstations, special hardware devices, testing labs, testing tools and equipment, team rooms, and videoconferencing equipment.
- Staff training needs.
- User documentation, including training materials, tutorials, reference manuals, and release notes.
- Support documentation, such as help desk resources and field maintenance and service information for hardware devices.
- Infrastructure changes needed in the operating environment.
- Requirements and procedures for releasing the product, installing it in the operating environment, configuring it, and testing the installation.
- Requirements and procedures for transitioning from an old system to a new one, such as data migration and conversion requirements, security setup, production cutover, and training to close skills gaps; these are sometimes called *transition requirements* (IIBA 2009).
- Product certification and compliance requirements.
- Revised policies, processes, organizational structures, and similar documents.
- Sourcing, acquisition, and licensing of third-party software and hardware components.
- Beta testing, manufacturing, packaging, marketing, and distribution requirements.
- Customer service-level agreements.
- Requirements for obtaining legal protection (patents, trademarks, or copyrights) for intellectual property related to the software.

Requirements development and management



Requirements Elicitation

Elicitation encompasses all of the activities involved with discovering requirements,
such as interviews, workshops, document analysis, prototyping, and others.

The key actions are:

- Identifying the product's expected user classes and other stakeholders.
- Understanding user tasks and goals and the business objectives with which those tasks align.
- Learning about the environment in which the new product will be used.
- Working with individuals who represent each user class to understand their functionality needs and their quality expectations.

Usage-centric or product-centric?

Requirements elicitation typically takes either a usage-centric or a product-centric approach, although other strategies also are possible. The usage-centric strategy emphasizes understanding and exploring user goals to derive the necessary system functionality. The product-centric approach focuses on defining features that you expect will lead to marketplace or business success. A risk with product-centric strategies is that you might implement features that don't get used much, even if they seemed like a good idea at the time. We recommend understanding business objectives and user goals first, then using that insight to determine the appropriate product features and characteristics.

Requirements Analysis

Analyzing requirements involves reaching a richer and more precise understanding of each requirement and representing sets of requirements in multiple ways.

Following are the principal activities:

- Analyzing the information received from users to distinguish their task goals from functional requirements, quality expectations, business rules, suggested solutions, and other information
- Decomposing high-level requirements into an appropriate level of detail
- Deriving functional requirements from other requirements information
- Understanding the relative importance of quality attributes
- Allocating requirements to software components defined in the system architecture
- Negotiating implementation priorities
- Identifying gaps in requirements or unnecessary requirements as they relate to the defined scope

Requirements Specification

Requirements specification involves representing and storing the collected requirements knowledge in a persistent and well-organized fashion.

The principal activity is:

- Translating the collected user needs into written requirements and diagrams suitable for comprehension, review, and use by their intended audiences.

Requirements Validation

Requirements validation confirms that you have the correct set of requirements information that will enable developers to build a solution that satisfies the business objectives.

The central activities are:

- Reviewing the documented requirements to correct any problems before the development group accepts them.
- Developing acceptance tests and criteria to confirm that a product based on the requirements would meet customer needs and achieve the business objectives.

Iteration

Iteration is a key to requirements development success. Plan for multiple cycles of exploring requirements, progressively refining high-level requirements into more precision and detail, and confirming correctness with users. This takes time and it can be frustrating. Nonetheless, it's an intrinsic aspect of dealing with the fuzzy uncertainty of defining a new software system.

Important You're never going to get perfect requirements. From a practical point of view, the goal of requirements development is to accumulate a shared understanding of requirements that is *good enough* to allow construction of the next portion of the product—be that 1 percent or 100 percent of the entire product—to proceed at an acceptable level of risk. The major risk is that of having to do excessive unplanned rework because the team didn't sufficiently understand the requirements for the next chunk of work before starting design and construction.

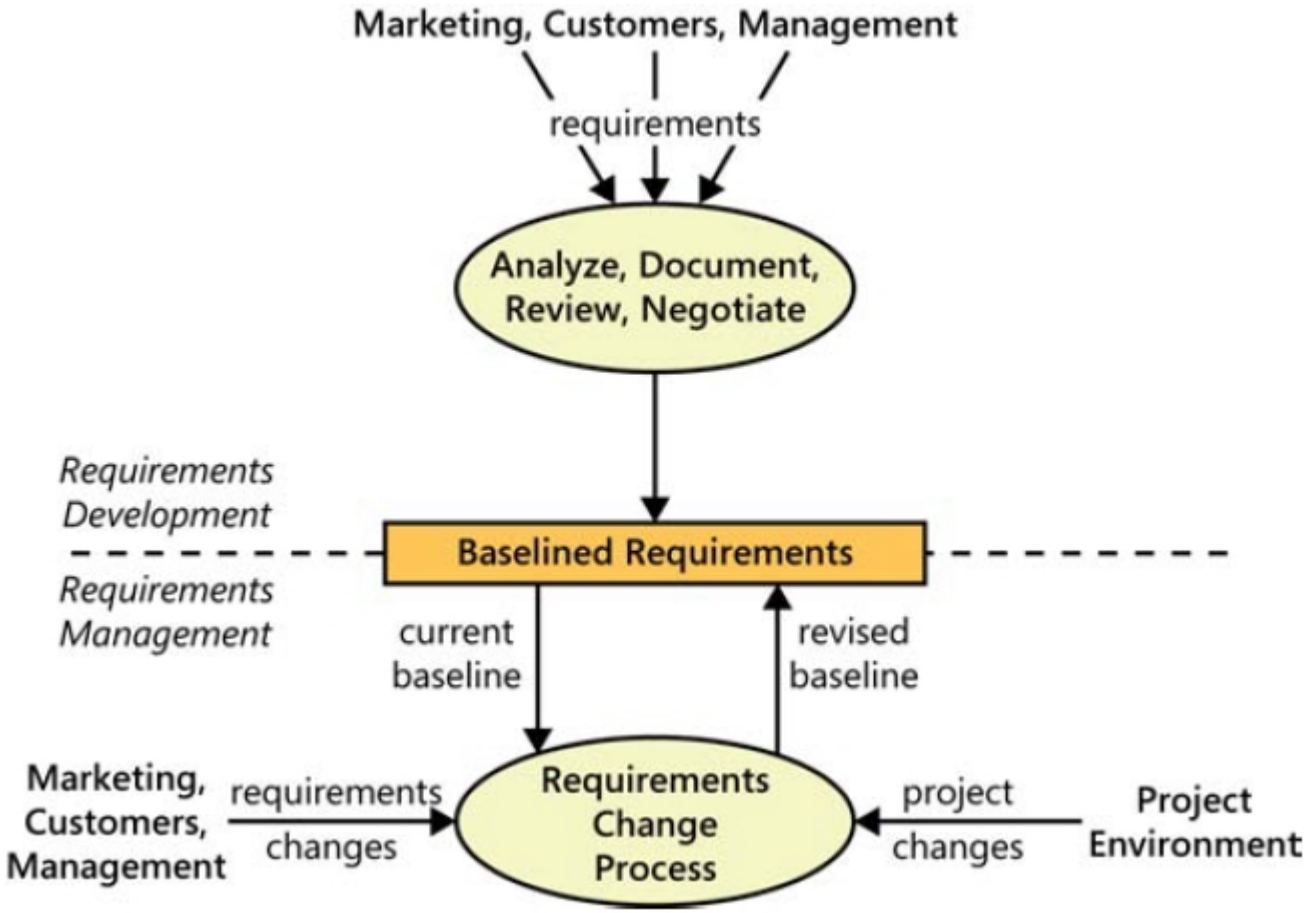
Requirements Management

Requirements management activities include the following:

- Defining the requirements baseline, a snapshot in time that represents an agreed-upon, reviewed, and approved set of functional and nonfunctional requirements, often for a specific product release or development iteration
- Evaluating the impact of proposed requirements changes and incorporating approved changes into the project in a controlled way
- Keeping project plans current with the requirements as they evolve
- Negotiating new commitments based on the estimated impact of requirements changes
- Defining the relationships and dependencies that exist between requirements
- Tracing individual requirements to their corresponding designs, source code, and tests
- Tracking requirements status and change activity throughout the project

The object of requirements management is not to stifle change or to make it difficult. It is to anticipate and accommodate the very real changes that you can always expect so as to minimize their disruptive impact on the project.

The boundary between requirements development and requirements management



When bad requirements happen to good people

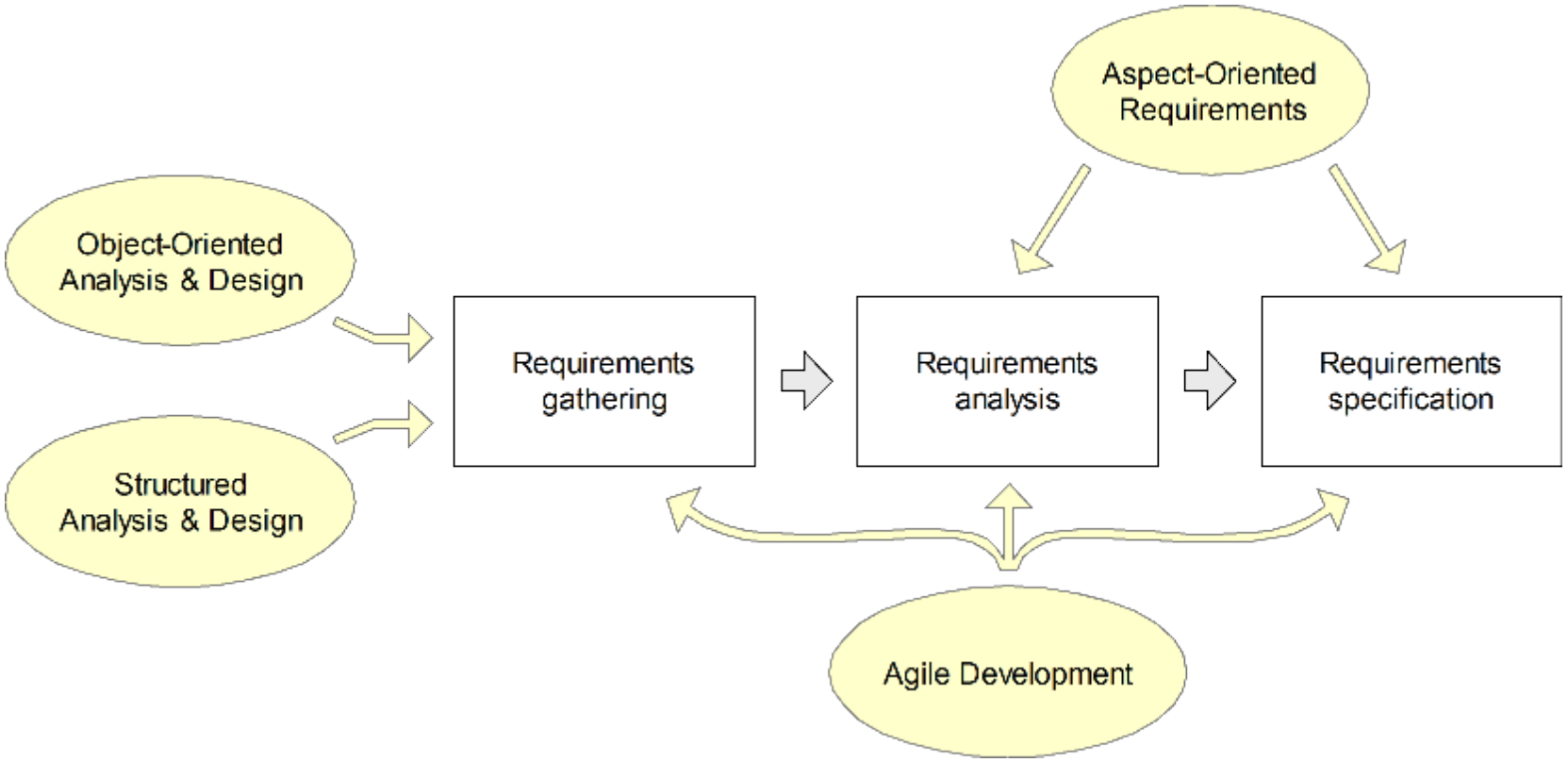
- Insufficient user involvement
- Inaccurate planning
- Creeping user requirements
- Ambiguous requirements
- Gold plating
- Overlooked stakeholders

Benefits from a high-quality requirements process

- ✓Fewer defects in requirements and in the delivered product.
- ✓Reduced development rework.
- ✓Faster development and delivery.
- ✓Fewer unnecessary and unused features.
- ✓Lower enhancement costs.
- ✓Fewer miscommunications.
- ✓Reduced scope creep.
- ✓Reduced project chaos.
- ✓Higher customer and team member satisfaction.
- ✓Products that do what they're supposed to do.

Even if you can't quantify all of these benefits, they are real.

Simplified Requirements Process



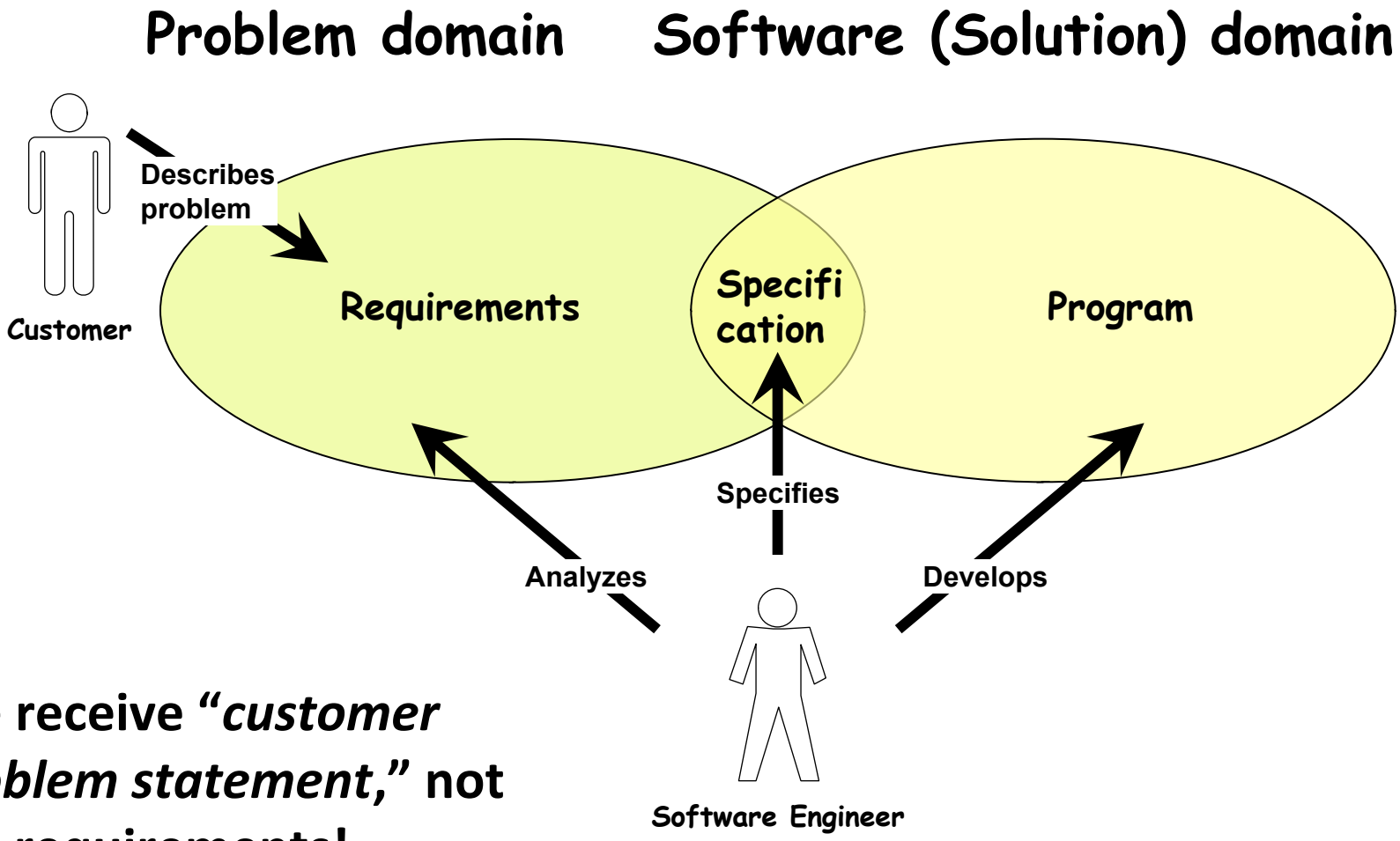
Requirements Engineering Components

- Requirements gathering
  - (a.k.a. “requirements elicitation”) helps the customer to define what is required: what is to be accomplished, how the system will fit into the needs of the business, and how the system will be used on a day-to-day basis
- Requirements analysis
  - refining and modifying the gathered requirements
- Requirements specification
  - documenting the system requirements in a semiformal or formal manner to ensure clarity, consistency, and completeness

Practical Requirements Engineering

- Test your idea in practice and use the result in further work, iterating through these creative and evaluative steps until a solution is reached
  - No one can know all the constraints for a solution before they go through the solving experience
- Define the criteria for measuring the success ("acceptance tests")
- Avoid *random* trial-and-error by relying on domain knowledge (from publications or customer expertise)

Requirements and Specification



We receive "*customer problem statement*," not the requirements!

Requirements Derivation

Detecting that a problem exists is different from *defining* the problem and its causes, and solution constraints.
Depending on the cause, the solution will be different.

Requirements are determined by:

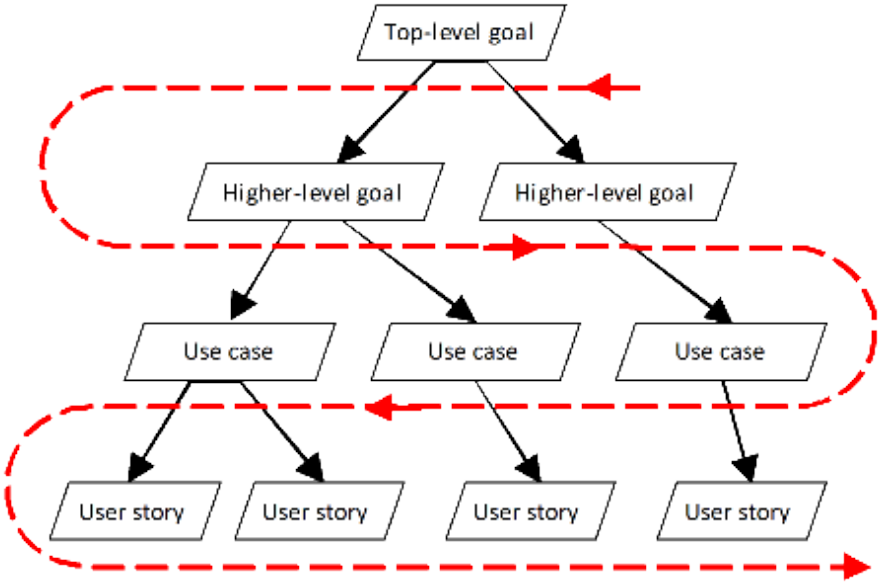
- Judgment about customer's business goals and obstacles that currently are hindering their achievement
- Conditions on solutions imposed by real-world constraints:
  - Physical
  - Social/Cultural
  - Legal
  - Financial
  - …
- Threats created by adversaries

➔ Requirements are **not** simply desires!

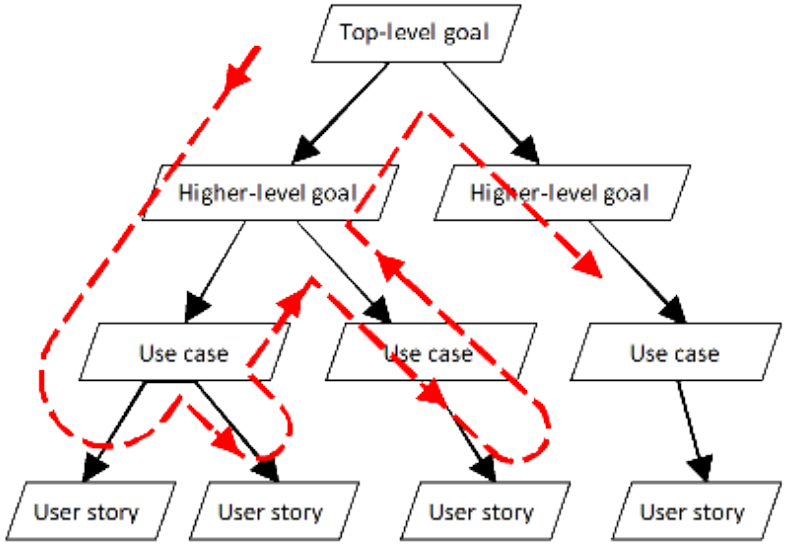
- Requirements are desires *adjusted to real-world constraints and threats*

Decomposition of Business Goals

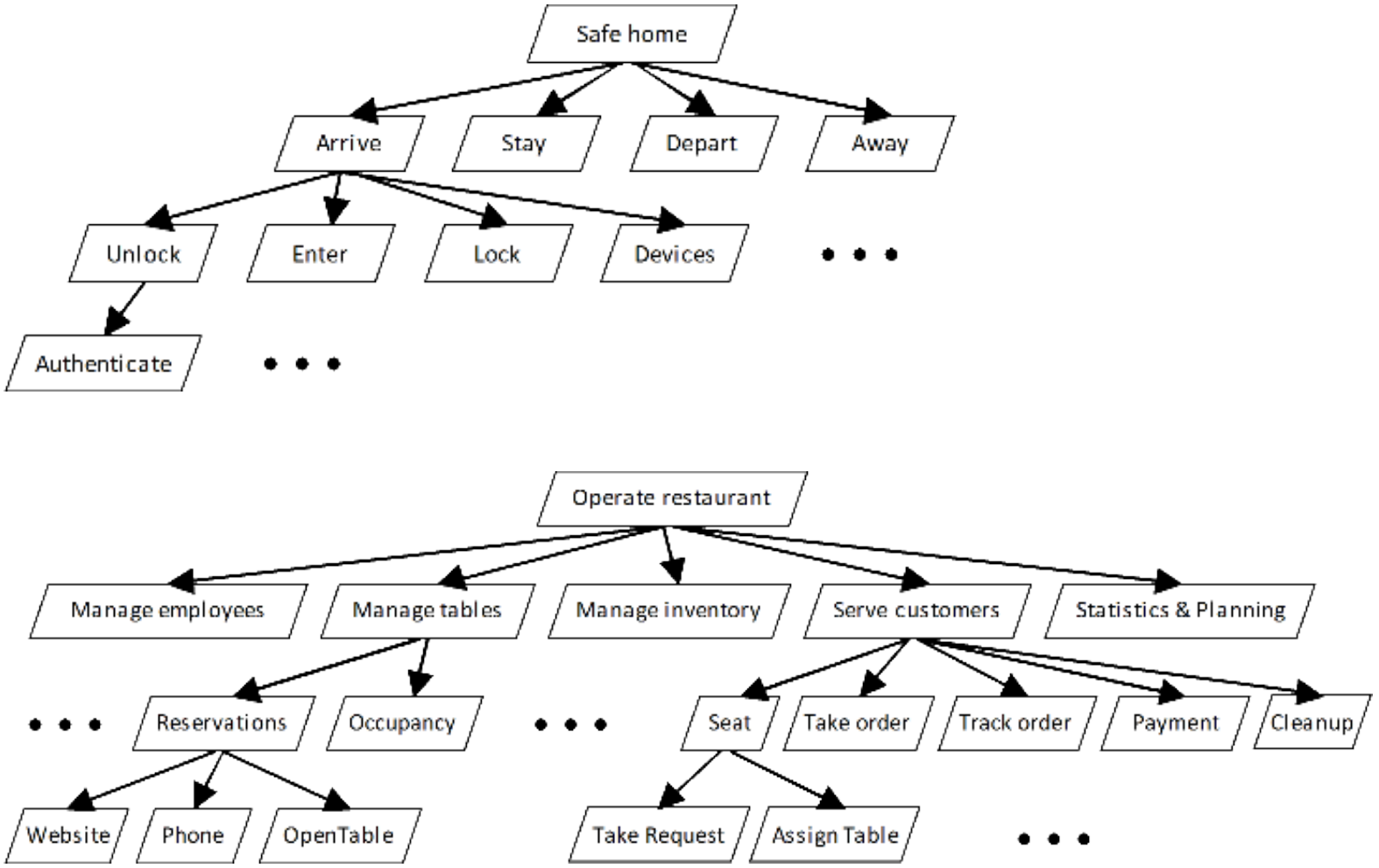
Breadth-first refinement



Depth-first refinement



Examples: Safe Home Access Restaurant Automation



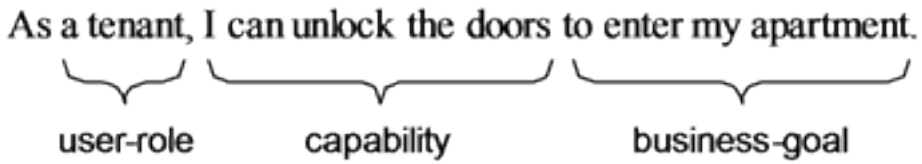
Problem Analysis Examples

- Traffic Monitoring:
  - Problem: User is delayed to work and needs help
  - Cause A: Traffic is unpredictable (predictably high if repeated delays?!)
  - Cause B: User is unfamiliar with the route
  - Cause C: User has a habit of starting late
- Restaurant Automation:
  - Problem: Restaurant is operating at a loss and customers are unhappy
  - Cause A: Staff is encumbered with poor communication and clerical duties, resulting in inefficiencies
  - Cause B: The menu does not match the likeliest customer's taste
  - Cause C: Staff is misbehaving with customers or mismanaging the food supplies
- Health Monitoring:
  - Problem: User is leading unhealthy lifestyle and experiencing health problems
  - Cause A: User is trying to be active but unsure about sufficient level of activity
  - Cause B: User is trying to be active but too busy with other duties
  - Cause C: User cannot find affordable solutions for active lifestyle
  - Cause D: User has poor sleep
  - Cause E: User is aware of issues but not motivated to act

Problem Example: Safe Home Access

- Problem detected:
  inconvenient physical keys or unwanted intrusion
  (plus: operating household devices and minimizing living expenses)
- Analysis of the Causes:
  - User forgets to lock the door or turn off the devices
  - User loses the physical key
  - Inability to track the history of accesses
  - Inability to remotely operate the lock and devices
  - Intruder gains access
- System Requirements: based on the selected causes

Requirements as User Stories

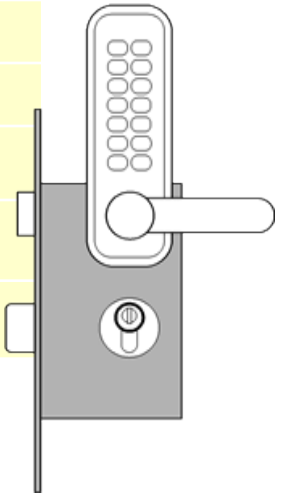


- Preferred tool in **agile methods**.
- Stated in terms of user's goals and capabilities instead of system features
- Written by the customer or user, not by the developer.
- The development effort to implement a story is estimated immediately
- Acceptance tests are written when the story is identified
- Requirements identified only for the next iteration, not for the whole project

Example User Story Requirements

Safe Home Access

Identifier	User Story	Size
REQ-1	As a user, I can be sure that the doors by default will be locked.	**4 points**
REQ-2	As a user, I will be able to unlock the doors using a valid key.	**7 points**
REQ-3	An intruder will not be able to unlock the doors by guessing a valid key; the system will block when it detects a "dictionary attack."	**7 points**
REQ-4	As a user, I can be sure that the doors will be automatically locked at all times.	**6 pts**
REQ-5	The door keypad will be backlit when dark for visibility.	**3 pts**
REQ-6	Anyone will be able to lock the doors on demand.	**3 pts**
REQ-7	As a user, I will be able to manage additional user accounts.	**10 pts**
REQ-8	As a user, I will be able to view the history of accesses to my home.	**6 pts**
REQ-9	As a user, I will be able to configure the preferences for how my household devices will be activated on my arrival.	**6 pts**



- Note no priorities for user stories
  - Story priority is given by its order of appearance on the to-do list
- Estimated size points (last column) will be described later (also see Chapter 1)
- Compare to IEEE-830 style requirements
  - https://en.wikipedia.org/wiki/Software_requirements_specification

Example User Story Requirements

Restaurant Automation

Identifier	User Story	Size
REQ-1	As a host, I can take a seating request including customer party information, place into the seating queue, and have a table assigned or waiting time estimated.	10 points
REQ-2	As a waiter, I can input customer's order.	5 points
REQ-3	As a waiter, I can add special instructions to an order at the customer's request.	2 points
REQ-4	As a waiter, I can notify the chef of the order without walking to the kitchen.	2 pts
REQ-5	As a waiter, I can view customer's bill and enter their payment information.	7 pts
REQ-6	As a waiter, I will be notified when an order has been completed.	2 pts
REQ-7	As a chef, I can see the queue of orders waiting to be prepared.	3 pts
REQ-8	As a chef, I can mark orders as "In Preparation" and "Complete".	2 pts
REQ-9	As a chef, I can modify the menu to make certain dishes available or unavailable if supplies are limited.	6 pts
REQ-10	As a chef, I can adjust and update the supply inventory to let the manager know of any lacking ingredients.	7 pts

- After requirements analysis:

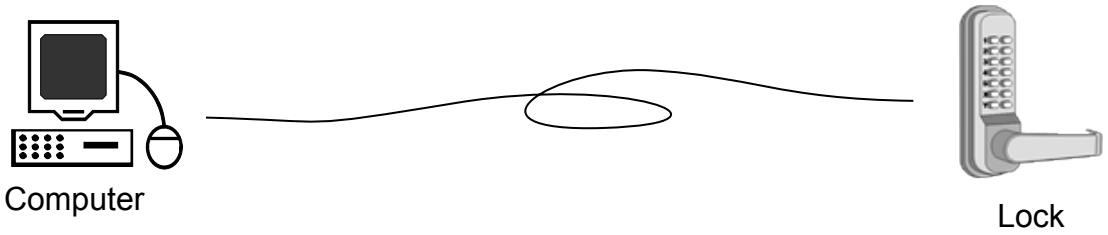
Identifier	User Story	Size
REQ-2a	As a waiter, I can input orders at different times for customers at the same table.	7 points
REQ-2b	As a waiter, I can input different courses at different times for the same customer.	5 pts
REQ-2c	As a waiter, I can input side plates after the main course is served.	2 pts

Requirements Analysis

- Requirement REQ-3 states that intruders will not be able to succeed with a "dictionary attack," but many details need to be considered and many parameters determined ("business policies")
  - What distinguishes user's mistakes from "dictionary attacks"
    - The number of allowed failed attempts, relative to a predefined threshold value
      - The threshold shall be small, say three ← business policy!
  - How is the mechanical lock related to the "blocked" state?
    - Can the user use the mechanical key when the system is "blocked"?
- Requirement REQ-5 states that the keypad should be backlit when dark
  - Is it cost-effective to detect darkness vs. keep it always lit?
- Etc.

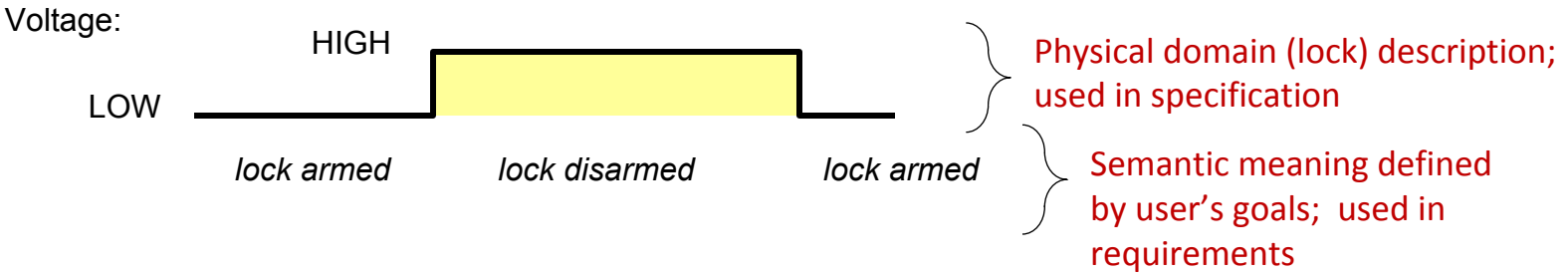
Requirements analysis should not be exhaustive,
but should neither be avoided.

Problem Domain: How Electronic Lock Works



We may need separate descriptions/models of door vs. lock.

Door state is what the user cares about; lock is one way of achieving it.



The behavior of the system-to-be determined not only by user's actions but also by the context ("situation").

E.g., what in case of power failure?

- By default armed
- By default disarmed (e.g., fire exit)

Analyst's Task: Three Descriptions

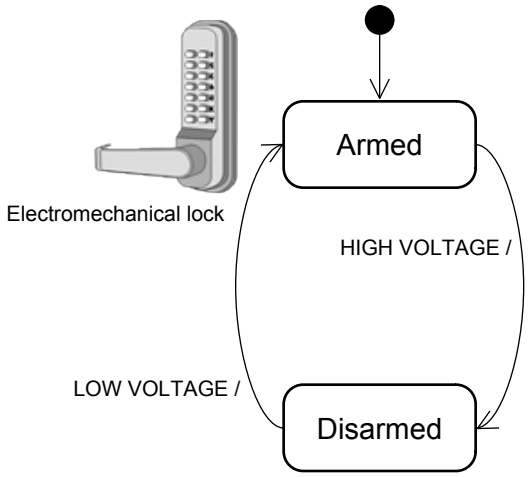
The ***requirement*** + The ***problem domain*** → The ***specification***

What user wants:

When valid keycode entered & Unlock pressed, open the lock;

Automatically lock after a period of time.

How problem domain behaves:



What software-to-be will do (at interface):

If entered number matches one of stored numbers **&** Button-1 pressed, put HIGH voltage on Output-port-1;

Start a timer countdown;

When the timer expires, put LOW voltage on Output-port-1.

Concern:

It is not obvious that this is the only or even "correct" solution to the requirement-posed problem.

Problem Frames tell us what each description should contain and how to verify the concern.

From Requirements to Business Policies

Explicit identification of **business policies** is important for two reasons:

1. Making the need for BP explicit allows involving other stakeholders, particularly the customer, in decision making about the BP solutions to adopt

2. Helps to anticipate potential future changes in the policies, so mechanisms can be implemented in the code that localize the future changes and allow quick substitution of implemented business policies

These issues too important to be left to the programmer to make ad-hoc decisions and hard-code them.

Requirements Analysis Activities

- Not only refinement of customer requirements, but also feasibility and how realistic
- Needs to identify the points where **business policies** need to be applied.
- Explicit identification of business policies (BP) is important for two reasons:
  1. Making the need for BP explicit allows involving other stakeholders in decision about the solutions to adopt—Helps to involve others, particularly the customer, in decision making about each policy to adopt
  2. Helps to anticipate potential future changes in the policies, so mechanisms can be implemented in the code that localize the future changes and allow quick substitution of business policies

Types of Requirements

- Functional Requirements
- Non-functional requirements (or quality requirements)
  - FURPS+
  - Functionality (security), Usability, Reliability, Performance , Supportability
- User interface requirements

FURPS+

- **F**unctional (features, capabilities, security)
- **U**sability (human factors, help, documents)
- **R**eliability (failures, recovery, predictable)
- **P**erformance (response, throughput, etc)
- **S**upportability (maintainability, configuration)
- **+** ancillary and sub-factors (next slide)

Ancillary and sub-factors

- Implementation (includes limitations)
- Interface
- Operations
- Packaging
- Legal Requirements

Functional Requirements

- Detailed in the Use Case Model and in the System Features list of the Vision artifact. They are specified in detail in Operation Contracts where necessary.

Non-functional requirements

- Often called the "-ilities" of a system; quality, reliability, usability, performance, etc.
- The glossary, data dictionary and supplemental specifications describe many non-functional requirements.
- In addition, architectural documents may have non-functional requirements.

Tools for Requirements Eng.

- Tools, such as user stories and use cases, used for:
  - Determining what exactly the user needs ("requirements analysis")
  - Writing a description of what system will do ("requirements specification")
- Difficult to use the same tool for different tasks (analysis vs. specification)

Acceptance Tests

- Each *requirement* describes for a given "situation" (i.e., system *inputs*), the *output* or behavior the system will produce
  - The "output" represents the user's need or business goal
- An **acceptance test** specifies a set of scenarios for determining whether the (part of the) system meets the customer requirements
- An **acceptance test *case*** specifies, for a given "situation" or "context" (defined by current system inputs), the output or behavior the system will produce in response
- We cannot ever guarantee 100% coverage of all usage scenarios, but *systematic approach* can increase the *expected* degree of coverage

[See examples in Appendix G of the TextA]

Project Estimation using User Story Points

- Similar to "hedge pruning points" in the first lecture
- Points assigned to individual user stories
- Total work size estimate:

$$\text{Total size} = \sum \text{points-for-story } \boldsymbol{i} \quad (i = 1..N)$$

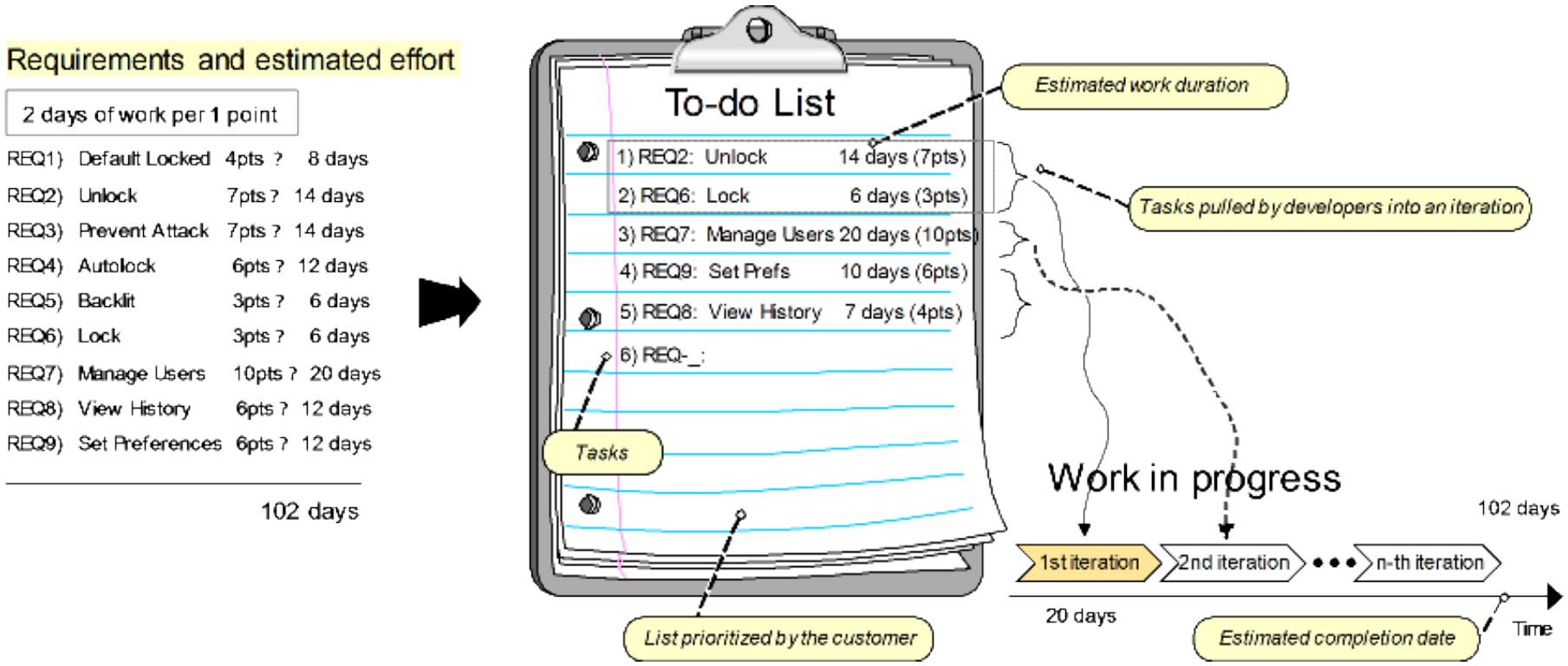
- Velocity (= productivity) estimated from experience
- Estimate the work duration

$$\text{Project duration} = \frac{\text{Path size}}{\text{Travel velocity}}$$

Example User Stories

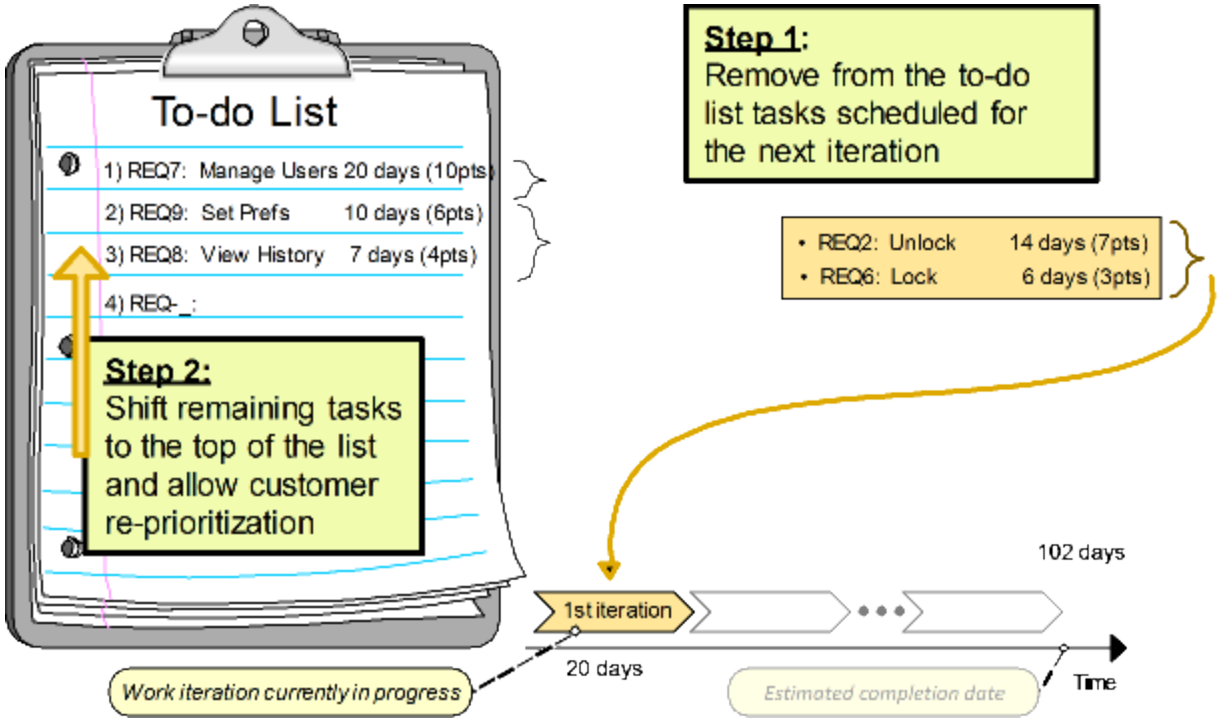
Identifier	User Story	Size
REQ-1	As an authorized user, I will be able to keep the doors by default always locked.	**4 points**
REQ-2	As an authorized user, I will be able to unlock the doors using a valid key.	**7 points**
REQ-3	An intruder will not be able to unlock the doors by guessing a valid key; the system will block upon a “dictionary attack.”	**7 points**
REQ-4	The door will be automatically locked after being open for a defined period of time.	**6 pts**
REQ-5	As a user, I will have the door keypad backlit when dark for visibility.	**3 pts**
REQ-6	Anyone will be able to lock the doors on demand.	**2 pts**
REQ-7	As a landlord, I will be able at runtime to manage user authorization status.	**10 pts**
REQ-8	As an authorized user, I will be able to view the history of accesses and investigate “suspicious” accesses.	**6 pts**
REQ-9	As an authorized user, I will be able to configure the preferences for activation of household devices.	**6 pts**

Agile Estimation of Project Effort



- Instead of assigning priorities, the customer creates an ordered list of user stories ➔ TO-DO LIST
- Developers simply remove the top list items and work on them in the next iteration ➔ IN-PROGRESS LIST

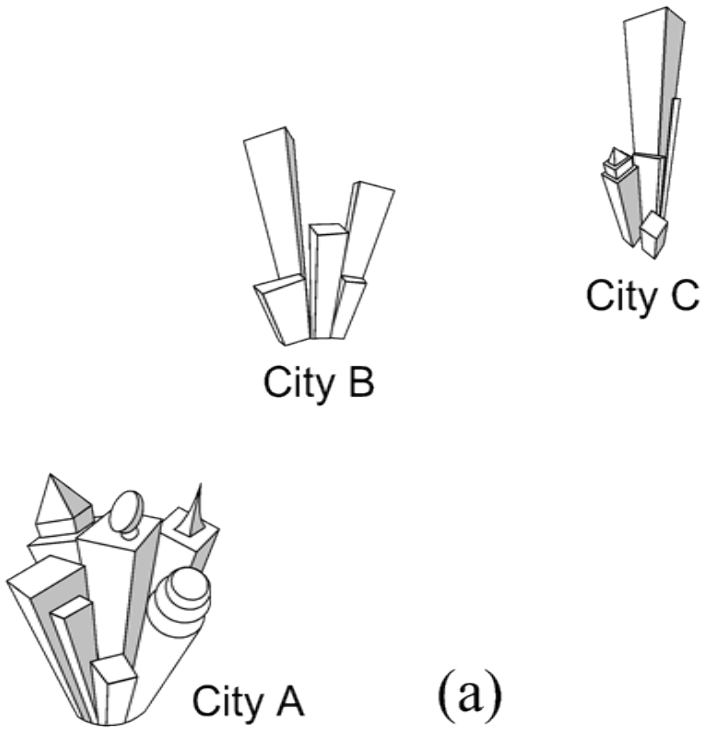
Tradeoff between Customer Flexibility and Developer Stability

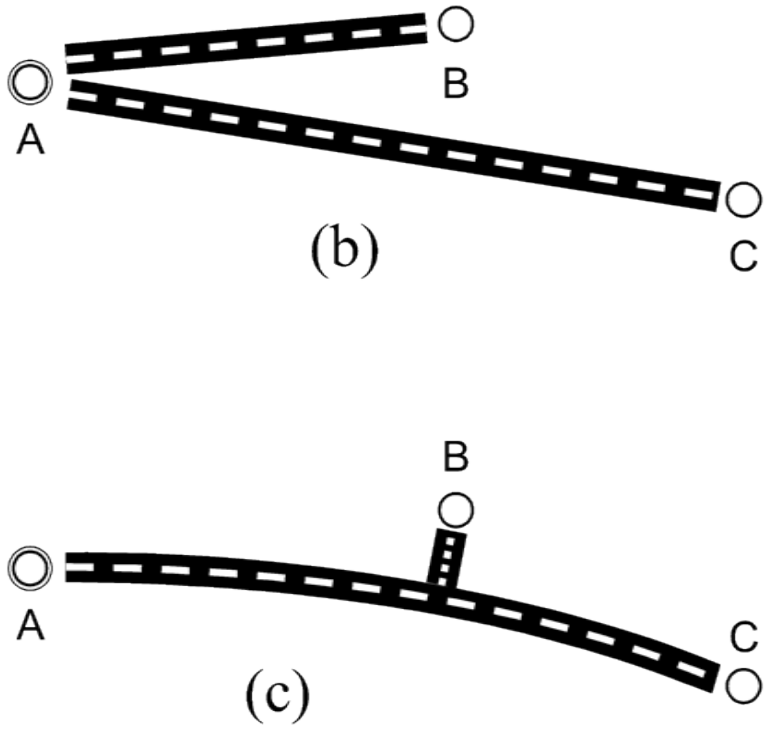


- Items pulled by developers into an iteration are not subject to further customer prioritization
- Developers have a **steady goal** until the end of the current iteration
- Customer has **flexibility** to change priorities in response to changing market forces

[Read more in Section 1.4.1 of TextA]

How To Combine the Part Sizes?

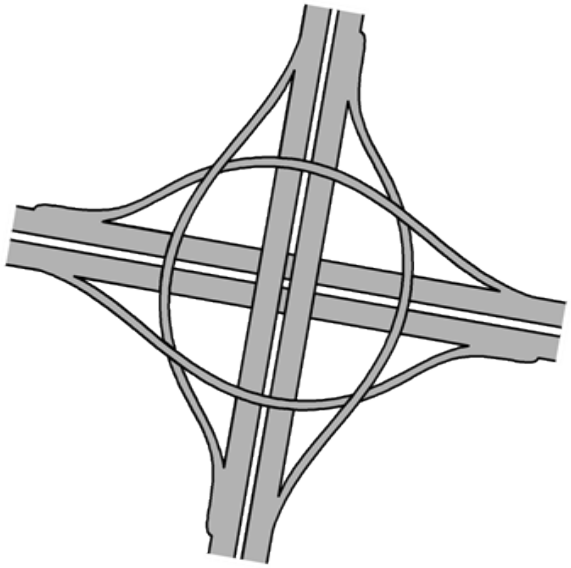




Costs are not always additive

But, solution (c) is not necessarily “cheaper” than (b) …

Additional Costs



Highway traffic-circle interchange

Traffic signs