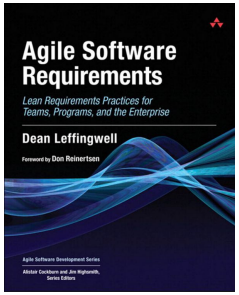
CE Department

Software Requirements Engineering

40688

These slides are designed to accompany Agile Software Requirements (2011) by Dean Leffingwell and support the university course Software Requirements Engineering, instructed by Mehran Rivadeh. Created and designed by Mahnaz Rasekhi.



Agile Software Requirements (2011)
Dean Leffingwell

Requirements Analysis Toolkit

Chapter 18

Mehran Rivadeh
mrivadeh@sharif.edu
Software Requirements Engineering
October 2025 - Fall 1404 - SUT

Contents

Introduction

Introduction

At The Team Level

- In smaller projects, communication among the stakeholders is not a big issue.
- The team usually has easy access to each other, and it's easy to resolve ambiguities and confusion.
  - Just lean over to the next workstation, and ask the product owner what they meant with a story!

Introduction

At The Program Level

- As projects grow in scope and size, the communication pathways become more and more complex.
- Opportunities for misunderstandings arise.
- Different means of communication become necessary.
- There are cases in which the ambiguity of imperfect requirements communication is simply not tolerable. For example:
  - When the stories deal with life-and-death issues.
  - When the erroneous behavior of a system could have extreme financial or legal consequences.

Introduction

- Communication has always been an imperfect vehicle.
- Misunderstandings abound.
- It's common to say or hear:
  - "This story is perfectly clear. Why don't you understand it?"
  - Indeed, it may be clear to the story writer, but others may not find it so obvious at all.

Introduction

- Sometimes far more precision is required.
  - Example: How exactly is that cardiac pacemaker algorithm supposed to work?
- When we are building complex systems, there are clearly times when we need alternate and far more precise communication mechanisms.
- We need to resolve ambiguity and build more assuredly safe and reliable systems.

Introduction

So

- If the description of the story is too complex for natural language and if the business cannot afford to have the specification misunderstood, the team should augment the story with a more precise specification method.
- There are a number of requirements analysis techniques, more technical methods for specifying system behavior that the team can use to resolve ambiguity and build more assuredly safe and reliable systems.

Introduction

- Such methods include the following:
  1. Activity diagrams (flowcharts)
  2. Sample reports
  3. Pseudocode
  4. Decision tables and decision trees
  5. Finite state machines
  6. Message sequence diagrams
  7. Entity-relationship diagrams
  8. Use cases

Introduction

- We provide a brief introduction to each so that you'll have a sense of what to use and when to use it.
- Where possible, only one or two of these technical methods should be used to augment natural-language stories for a system.
- This simplifies the nontechnical reviewer's task of reading and understanding these special elements.
- If all the systems developed by an organization fall into one application domain, such as telephone switching systems, perhaps the same technical method can be used for all the systems.
- But in most organizations, it's unrealistic to mandate a single technique for all stories in all systems.
- Story writers and analysts need to select an approach that best suits the individual situation and help the users and reviewers understand how the technique expresses system behavior.

Contents

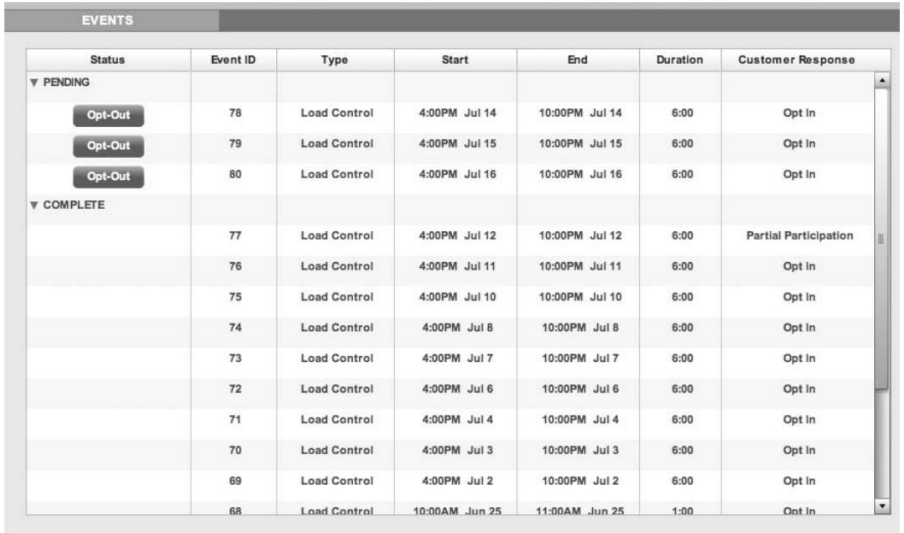
1. Sample Report

Sample Report

- Users often don’t know what they want to see in a report or other output until they see the report.
- “What is it that they really want to see, and how do they want to see it?”
- If the system is algorithmically intensive and the user cannot evaluate the report format without some real data and results, the team may need to invest some time in producing some real data.
- A sample report format with mock data, generated by any number of desktop tools, might be all that is necessary to resolve most of the ambiguity.

Sample Report

- Spikes are frequently invoked as research items to develop and validate sample reports with the product owner and users.



EVENTS

Status	Event ID	Type	Start	End	Duration	Customer Response
▼ PENDING						
Opt-Out	78	Load Control	4:00PM Jul 14	10:00PM Jul 14	6:00	Opt In
Opt-Out	79	Load Control	4:00PM Jul 15	10:00PM Jul 15	6:00	Opt In
Opt-Out	80	Load Control	4:00PM Jul 16	10:00PM Jul 16	6:00	Opt In
▼ COMPLETE						
	77	Load Control	4:00PM Jul 12	10:00PM Jul 12	6:00	Partial Participation
	76	Load Control	4:00PM Jul 11	10:00PM Jul 11	6:00	Opt In
	75	Load Control	4:00PM Jul 10	10:00PM Jul 10	6:00	Opt In
	74	Load Control	4:00PM Jul 8	10:00PM Jul 8	6:00	Opt In
	73	Load Control	4:00PM Jul 7	10:00PM Jul 7	6:00	Opt In
	72	Load Control	4:00PM Jul 6	10:00PM Jul 6	6:00	Opt In
	71	Load Control	4:00PM Jul 4	10:00PM Jul 4	6:00	Opt In
	70	Load Control	4:00PM Jul 3	10:00PM Jul 3	6:00	Opt In
	69	Load Control	4:00PM Jul 2	10:00PM Jul 2	6:00	Opt In
	68	Load Control	10:00AM Jun 25	11:00AM Jun 25	1:00	Opt In

Contents

2. Pseudocode

Pseudocode

Pseudocode combines the informality of natural language with the strict syntax and control structures of a programming language.

The Extreme Form

- Imperative sentences with a single verb and a single object
- A limited set, typically not more than 40 to 50, of “action-oriented” verbs from which the sentences must be constructed
- Decisions represented with a formal IF-ELSE-ENDIF structure
- Iterative activities represented with DO-WHILE or FOR-NEXT structures

Pseudocode

Example:

- Pseudocode specification of an algorithm for calculating deferred-service revenue earned within a given month in a business application.

```
Set SUM(x)=0
FOR each customer X
        IF customer purchased paid support
              AND((Current month)>=(2 months after ship date))
              AND((Current month)<=(14 months after ship date))
        THEN Sum(X)=Sum(X)+(amount customer paid)/12
END
```

Pseudocode

- The text of the pseudocode is indented in an outline-style format in order to show “blocks” of logic.
- The combination of the syntax restrictions and the format and layout of the text greatly reduces the ambiguity.
- A nonprogramming person can read and understand the story’s function.

```
Set SUM(x)=0
FOR each customer X
        IF customer purchased paid support
              AND((Current month)>=(2 months after ship date))
              AND((Current month)<=(14 months after ship date))
        THEN Sum(X)=Sum(X)+(amount customer paid)/12
END
```

Contents

3. Decision Table And Decision Trees

Decision Table And Decision Tree

- ➔ It's common to see a story that deals with a combination of inputs.
- ➔ Different combinations of those inputs lead to different actions or outputs.

For example:

- ➔ We have a system with three inputs: A, B, and C
- ➔ We see a story that starts with a pseudocode-like statement:
  - ◆ "If A is true, then if B is also true, do action X, unless C is true, in which case the required action is Y."
- ➔ The combination of IF-THEN-ELSE clauses quickly becomes tangled, especially if it involves nested IFs.

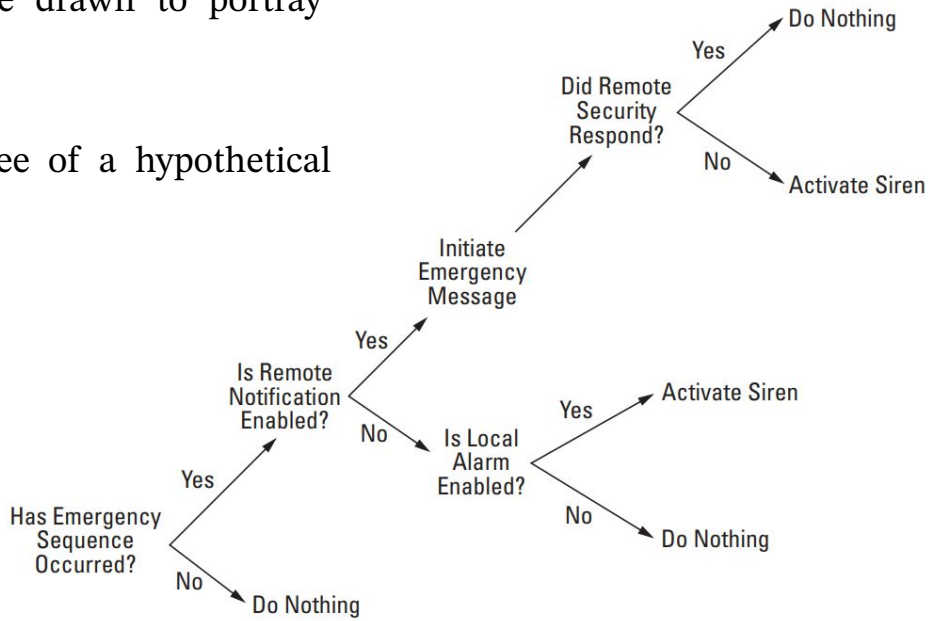
Decision Table And Decision Tree

- Typically, nontechnical users are not sure that they understand any of it.
- Nobody is sure whether all the possible combinations and permutations of A, B, and C have been covered.
- The solution in this case is to enumerate all the combinations of inputs and to describe each one explicitly in a decision table.

		Rules							
Conditions	Printer does not print.	Y	Y	Y	Y	N	N	N	N
	A red light is flashing.	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized.	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable.			X					
	Check the printer-computer cable.	X		X					
	Ensure printer software is installed.	X		X		X		X	
	Check/replace ink.	X	X			X	X		
	Check for paper jam.		X		X				

Decision Table And Decision Tree

- Alternatively, a decision tree can be drawn to portray decisions in a more graphical form.
- This is an example of a decision tree of a hypothetical emergency sequence.



Contents

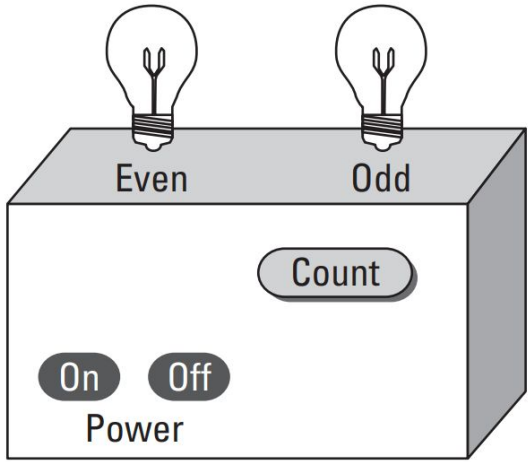
4. Finite State Machines

Finite State Machine

- In some cases, it's convenient to model a complex system as a "hypothetical machine that can be in only one of a given number of 'states' at any specific time" [Davis 1993].
- In response to an input, such as data entry from the user or an input from an external device, the machine changes its state and then generates an output or carries out an action.
- Both the output and the next state can be determined solely on the basis of understanding the current state and the event that caused the transition.
- In that way, a system's behavior can be said to be deterministic; we can mathematically determine every possible state and, therefore, the outputs of the system, based on any set of inputs provided.

Finite State Machine

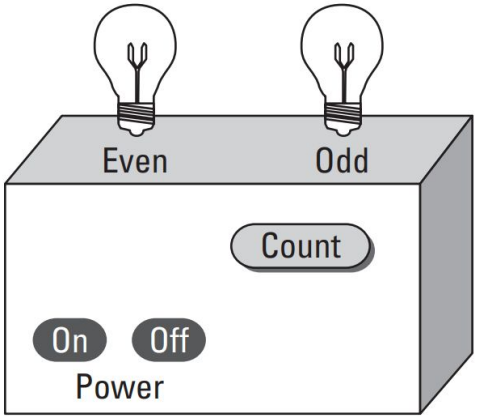
- Let's suppose that we have a light box with
  - Two lights (Even and Odd)
  - Three buttons:
    - On
    - Off
    - Count



Finite State Machine

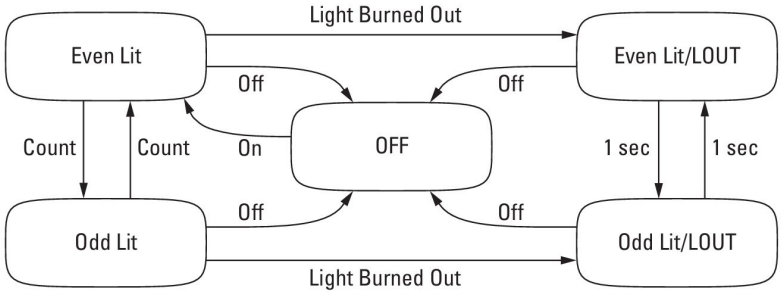
The Desired Story

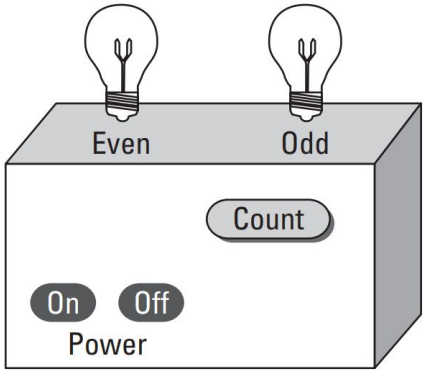
- After On is pushed but before Off is pushed, system is termed “powered on.”
- After Off is pushed but before On is pushed, system is termed “powered off,” and no lights shall be lit.
- Since the most recent On press, if Count has been pressed an odd number of times, Odd shall be lit.
- Since the most recent On press, if Count has been pressed an even number of times, Even shall be lit.
- If either light burns out, the other light shall flash every one second.



Finite State Machine

State Transition Machine



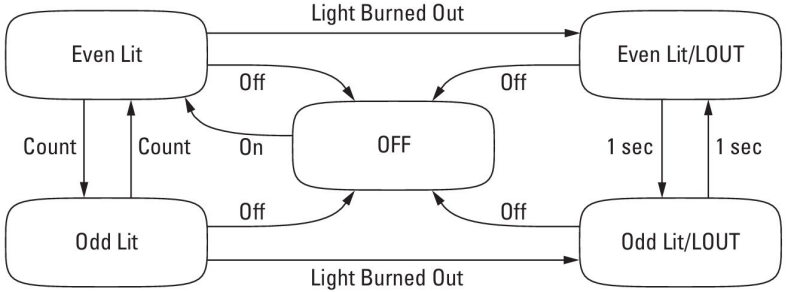


Finite State Machine

State Transition Machine (Cont.)

In this notation:

- ➔ The boxes represent the state the device is in.
- ➔ The arrows represent actions that transition the device to alternative states.

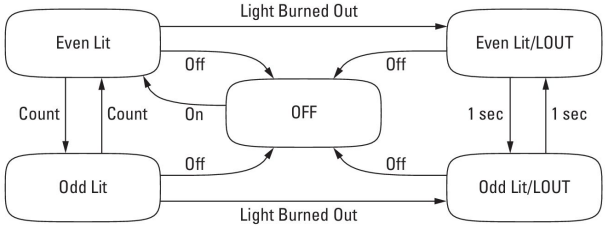


Finite State Machine

State Transition Machine (Cont.)

The Desired Story

- If either light burns out, the other light shall flash every one second.
  - It is ambiguous.
- The state transition diagram in the Figure is not ambiguous.
- It illustrates exactly what the product owner desired.

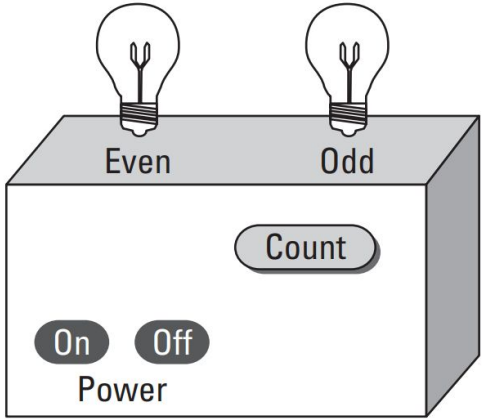


- If a bulb burns out, the device alternates between attempting to light the Even light and attempting to light the Odd light, each for a period of one second.

Finite State Machine

The State Transition Matrix

- There are additional ambiguities in our attempt to understand the behavior of the device.
  - What happens if the user presses the On switch and the device is already on?
    - Answer: Nothing.
  - What happens if both bulbs are burned out?
    - Answer: The device powers itself off.



Finite State Machine

The State Transition Matrix (Cont.)

- A more precise form of representing an FSM is **the state transition matrix**.
- It is represented as a table that shows:
  - Every possible state the device can be in
  - The output of the system for each state
  - The effect of every possible stimulus or event on every possible state.
- This ensures a higher degree of specificity, because every state and the effect of every possible event must be represented in the table.

Finite State Machine

State	Event					Output
	On Press	Off Press	Count Press	Bulb Burns Out	Every Second	
Off	Even Lit	—	—	—	—	Both Off
Even Lit	—	Off	Odd Lit	LO/Even Lit	—	Even Lit
Odd Lit	—	Off	Even Lit	LO/Odd Lit	—	Odd Lit
Light Out/Even Lit	—	Off	—	Off	LO/Odd Lit	Even Lit
Light Out/Odd Lit	—	Off	—	Off	LO/Even Lit	Odd Lit

Figure 18–8 Example of a state transition matrix

Finite State Machine

- FSMs are popular for certain categories of systems programming applications:
  - Message-switching systems
  - Operating systems
  - Process control systems
- FSMs also provide a rigorous way to describe the interaction between an external human user and a system (consider, for example, the interaction between a bank customer and an automated teller machine when the customer wants to withdraw money).

Finite State Machine

- ➔ However, FSMs can become unwieldy, particularly if we need to represent the system's behavior as a function of several inputs.
- ➔ In such cases, the required system behavior is typically a function of all current conditions and stimuli rather than the current stimulus or a history of stimuli.

Contents

5. Manage Sequence Diagrams

Message Sequence Diagram

- A message sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order.
- Message sequence diagrams (MSDs) are a convenient way to express a transactional relationship between two or more parties.
- Typically, MSDs are used to express relationships such as “A sends this message to B. B responds with this message to A.”

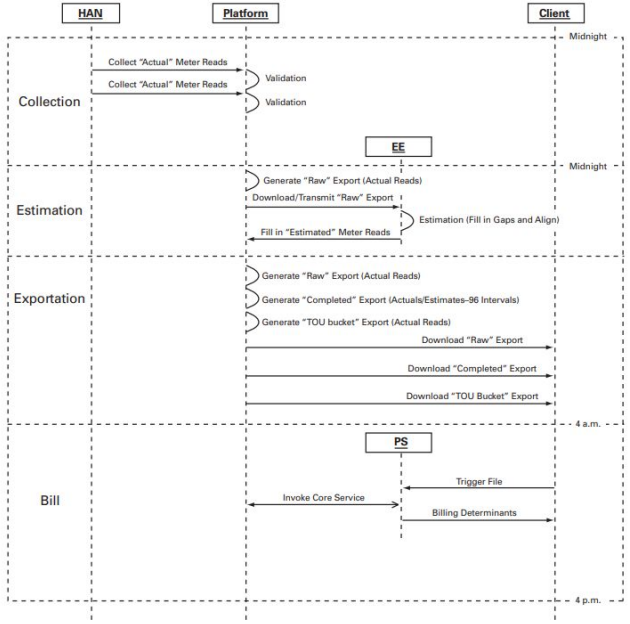
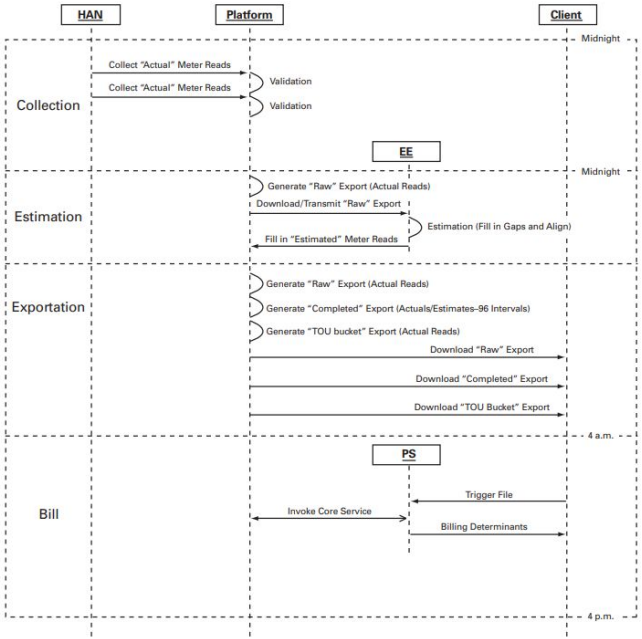


Figure 18–9 Case study example message sequence diagram

Message Sequence Diagram

- MSDs identify the interested parties (subsystems in this case) across the top of the diagram and the interactions proceed down the diagram as time evolves.

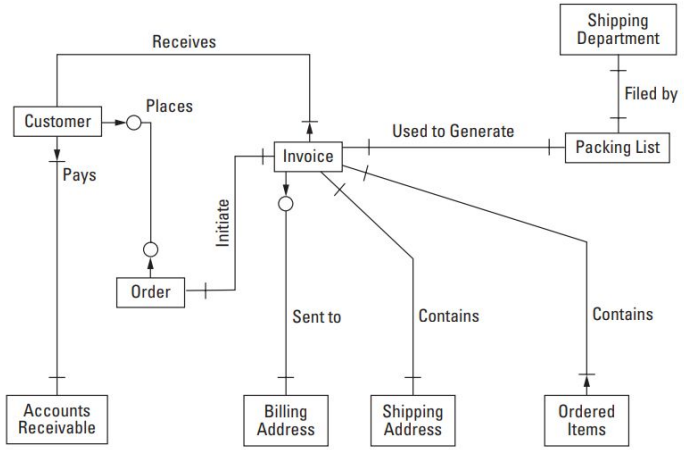


Contents

6. Entity-Relationship Diagrams

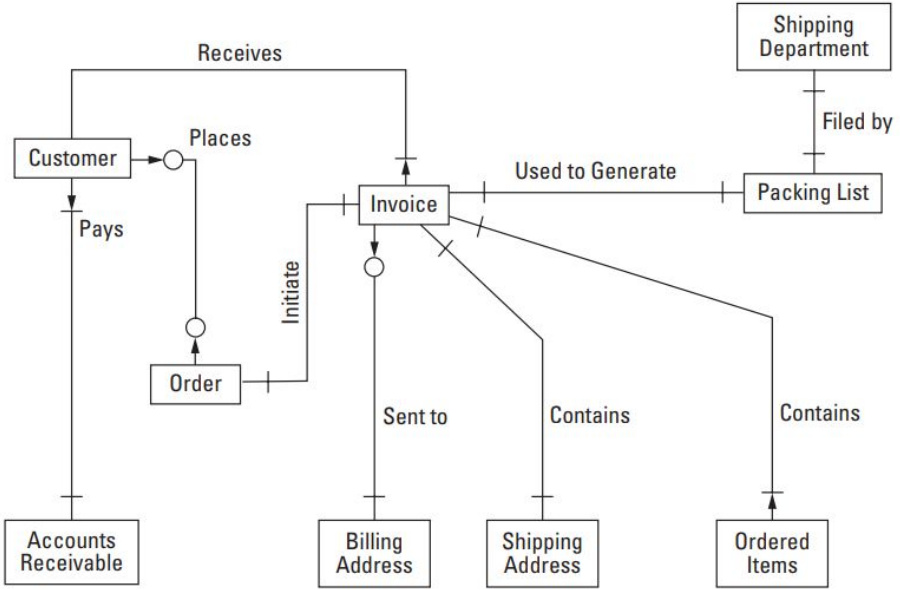
Entity-Relationship Diagram

- If the stories within a set involve a description of the structure and relationships among data within the system, it's often convenient to represent that information in an entity-relationship diagram (ERD).



Entity-Relationship Diagram

Example of an entity-relationship diagram



Entity-Relationship Diagram

- Note that the ERD provides a high-level “architectural” view of the data represented by customers, invoices, packing lists, and so on; it would be further augmented with appropriate details about the required information to describe a customer.
- The ERD does correctly focus on the external behaviors of the system and allows us to define such questions as “Can there be more than one billing address per invoice?” Answer: No.
- Although an ERD is a capable modeling technique, it has the potential disadvantage of being difficult for a nontechnical reader to understand.

Contents

7. Use-Case Modeling

Use-Case Modeling

- Use cases are a traditional way to express system behavior in complex systems.
- Use cases are the primary means to represent requirements with the UML.
- They are well described there as well as in a variety of texts on the subject.
- Use cases can be used for both specification and analysis.
- They are especially useful when the system of interest is in turn composed of other subsystems.

Conclusion

- Agile development avoids big, up-front design (BUFD) and analysis wherever possible.
- However, your system still has to work and deliver the requisite reliability.
- When user stories and natural language aren't good enough, your team will need to apply more technical methods to reduce the risk of misunderstanding and to provide additional safety, security, and reliability for your system.

Conclusion

- Introduced specification techniques can reduce ambiguity in specifying system behavior.
- These technical methods should be used sparingly, and common sense should guide the decision as to which formal technique will be used in a project.
- If you're building a pacemaker or nuclear reactor control system, perhaps every aspect of the system is critical.
- In most systems, however, it's unlikely that more than 10 percent of the stories will require this degree of rigor.
- Choose the method that suits your team best, and apply it only where it is really needed.

End of Chapter 18

Contributions

- Author of Reference Book: **Dean Leffingwell**
- Course Instructor: **Mehran Rivadeh**
- Slide Creator: **Mahnaz Rasekhi**
  - These slides are primarily based on Agile Software Requirements by Dean Leffingwell, with occasional adaptations to enhance clarity and engagement.