Advanced Software Engineering
(CS6401)

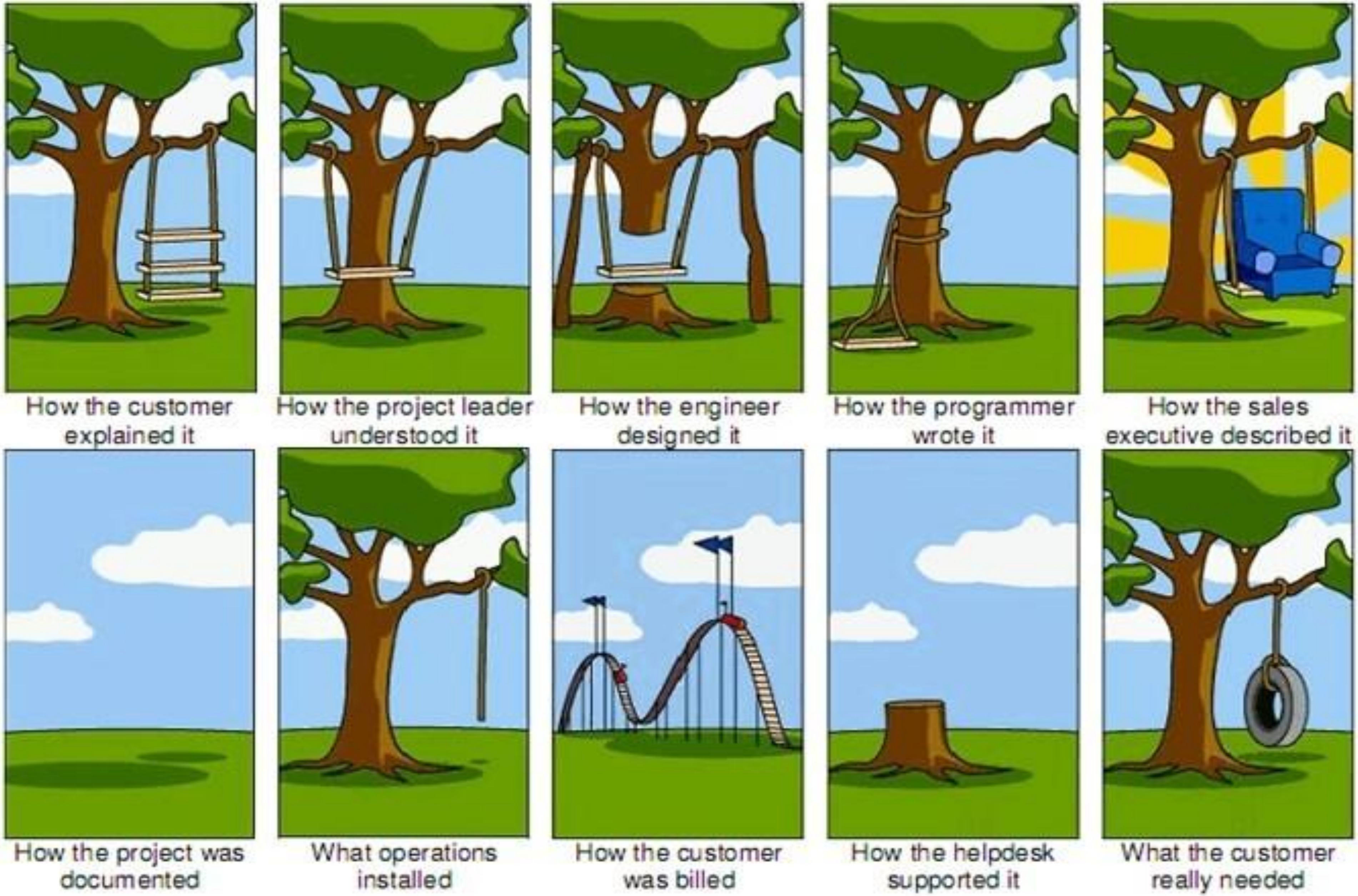
Autumn Semester (2022-2023)

Dr. Judhistir Mahapatro
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Topics covered in Previous Lecture:

- Requirement gathering and analysis
- Requirement engineering
- SRS

Formal Specification


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

- **Source of this image : https://www.businessballs.com/amusement-stress-relief/tree-swing-cartoons-new-versions/**

Formal methods

- Formal specification is part of a more general collection of techniques that are known as formal methods.
- These are all based on mathematical representation and analysis of software.

Formal methods

- Formal methods include
  - Formal specification
  - Specification analysis and proof (specification is checked for consistency)

Formal methods (Cont..)

- Formal methods include
  - Transformational development (if there were any means to take the requirements and verify the requirements and transform the requirement automatically into the code, then that would ensure that it will retain the continuity between various phases of the development cycle of the software.)
    - specification to piece of code
    - Persons reading the document and making analysis can have different interpretation, therefore, there will be human errors
    - If all of them can be automated and machine can do finally produce the code and which can then be verified by the human

Formal methods (Cont..)

- Formal methods include
  - Program verification
    - checking or testing to make sure that we have built the desired one,
    - it can also be automated,
    - the test case can be generated and applied on final production of the end product to verify the system what we are intended to.

Acceptance of formal methods

- Formal methods have not become mainstream software development techniques as was once predicted
  - Other SE techniques have been successful at increasing system quality. Hence, need for this has been reduced.

  - Market changes have made time-to-market the key factor. Formal methods do not reduce time-to-market because want to make the system without bugs at first go.

  - The scope of it is limited.
    - They are not well-suited to specifying and analyzing user interfaces and user interaction (that is, the front end GUI based system), but pretty good at backend system.

  - Formal methods are hard to scale up to large systems

Specification in the software process

- Specification and design are inextricably intermingled.
- Architectural design is essential to structure a specification.
- Formal specifications are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

Specification in the software process

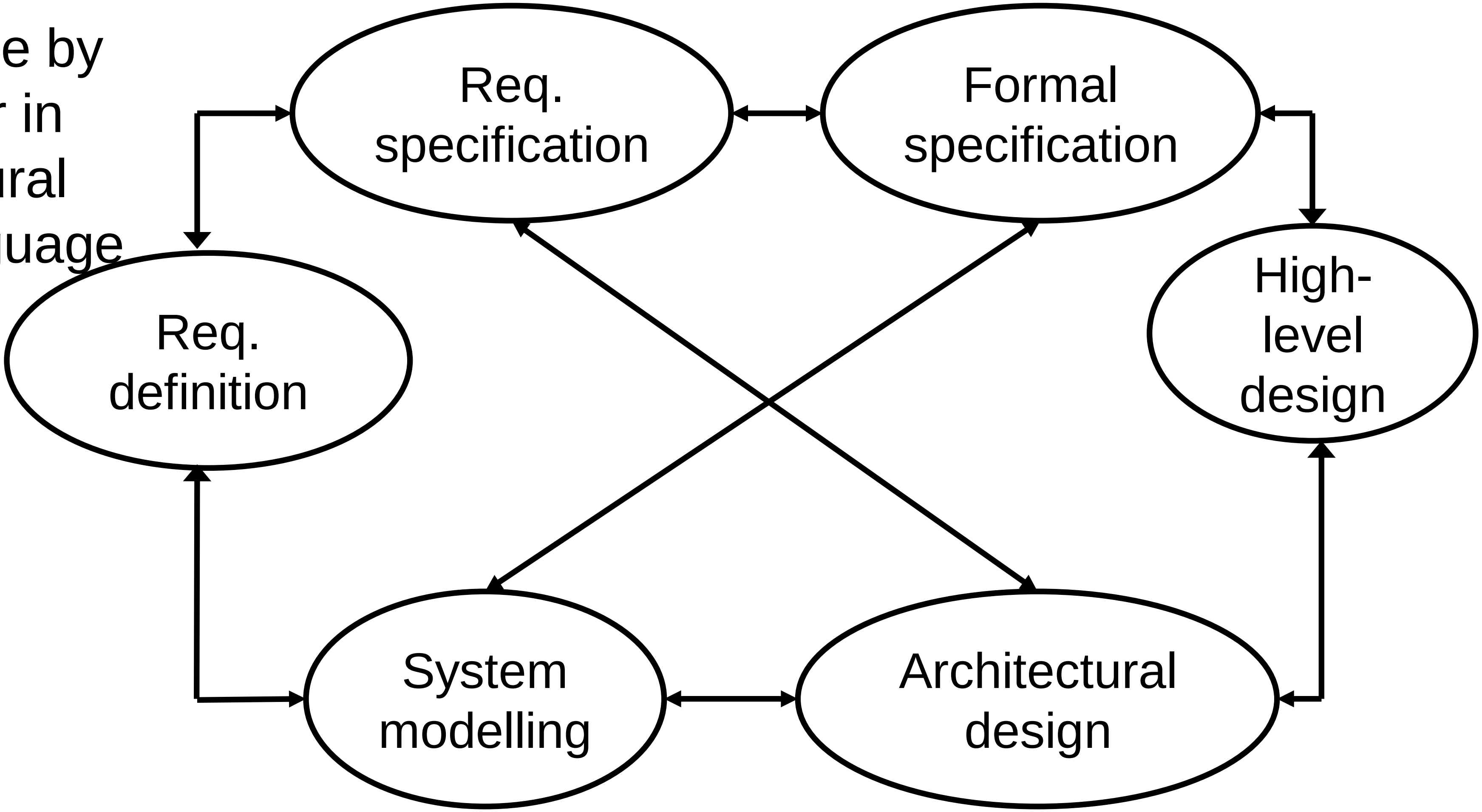
- Requirements definition can not be avoided as it is in the natural language.

- Requirements specification can be avoided and straight-away we can jump into the formal specification step. (Generally it is not recommended)

- High-level design can be generated once we get into the formal specification and different system models driven off from the system specification
  - For example, UML model can be generated from formal specification and test models can also be generated.

Specification in the software process

Done by user in natural language



Specification techniques

- Algebraic approach
  - The system is specified in terms of its operations and their relationships

- Model-based approach
  - The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences.
  - Operations are defined by modifications to the system's state.

Formal specification languages

	Sequential	Concurrent
Algebraic	Larch (Guttag et. al) OBJ (Futatsugi et. al)	Lotos (Bolognesi)
Model-Based	Z (Spivey et. al) VDM (Jones et. al)	CSP (Hoare) Petri Nets(Peterson)

- Concurrency introduces the notion that the multiple modules working at the same time.

Use of formal methods

- Formal methods have limited practical applicability

- Their principal benefits are in reducing the number of errors in systems so their main area of applicability is mission critical systems.
  - For example, software that runs in Avionics and large PLANES

- In this area, the use of formal methods is most likely to be cost-effective.

Use of formal specification (Cont..)

- Formal specification involves investing more effort in the early phases of software development
  - Taking much more time for modeling mathematically
    - but it ease out the later phases
  - Transformational developments is the focus of formal methods whereas manual or man based development is the focus of other software engineering practices.

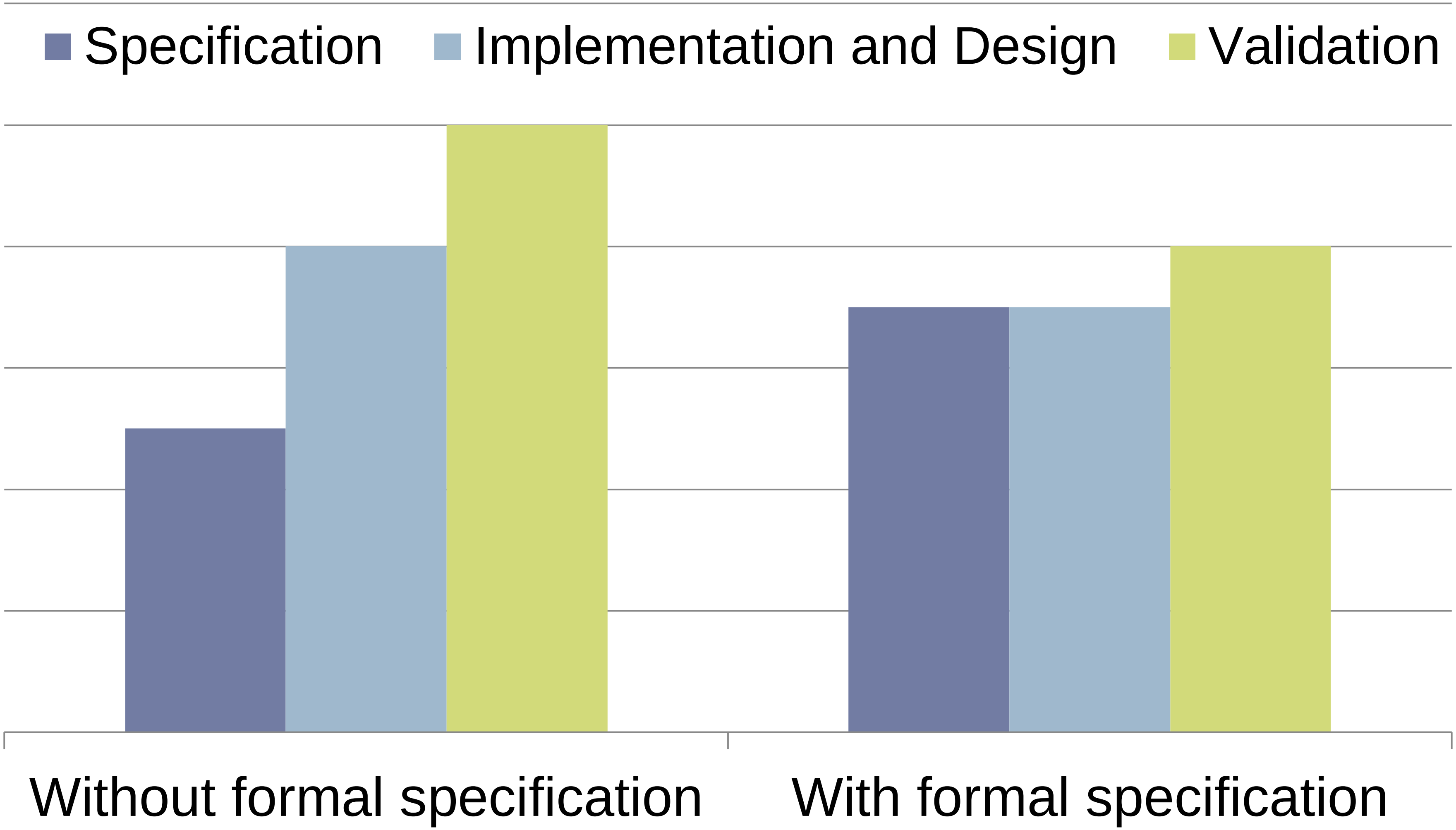
- This reduces requirements errors as it forces a detailed analysis of the requirements

Use of formal specification (Cont..)

- Incompleteness and inconsistencies can be discovered and resolved.

- Hence, savings as made as the amount of rework due to requirements problems is reduced.

Development costs with formal specification



Properties of Formal Specification

- Complete
  - specification has to be complete and the user definition document will be carried over appropriately
- Consistent
  - There should not be any ambiguity in the requirement
  - Example:

    “one requirement is that fewer chips has to be used that is more chips need to be put in one chip which increases power consumption and the power consumption of each chip should be lower”- inconsistency.

Properties of Formal Specification

- Concise
  - mid-sized projects even run into 100 of pages - verbose
- Unambiguous
- Executable (basis for transformational developments)
  - Z, VDM, Larch are directly executable that means that designs can be generated from these specification documents,
  - and then designs can be turned into codes automatically,
  - and code content can be directly executable if the designer gives green signal.

Properties of Formal Specification

- Reusable
  - for example, response time of a module that is a non-functional requirements (module calling another module should not take more than 50 millisecond) and this specification can be applicable in variety of applications (for example, applicable in the construction of website).
  - This piece of specification written for some system can be allowed to reuse in the development of other system.

Properties of Formal Specification

- Provides a common language for communication between clients, designers and implementors
- Has its basis in discrete mathematics
  - Set Theory
  - Boolean Logic
  - Predicate Logic

Library Example – Informal statement

- A book can either be in the stacks, on reserve, or loaned out.
- If a book is in the stacks or on reserve, then it can be requested.
- We want to
  1. formalize the concepts and the statements
  2. prove some theorems to gain confidence that the specification is correct

Library Example – Formalization -1

- Lets first formalize some concepts
  - S: the book is in the stacks
  - R: the book is on reserve
  - L: the book is on loan
  - Q: the book is requested

Library Example – Formalization - 2

- A book can either be in the stacks, on reserve, or loaned out
  - $S \wedge \neg(R \vee L)$
  - $R \wedge \neg(S \vee L)$
  - $L \wedge \neg(S \vee R)$
- If a book is in the stacks or on reserve, then it can be requested
  - $Q \Rightarrow (S \vee R)$

Library Example – Prove Correctness

- A theorem to help validate the specification.
- A book on loan cant be requested.
  - L => ¬Q
- See if the specification matches our understanding (and vice versa)
- Proof by contradiction
  - We start out by making one assumption: that the statement is false
  - Derive a contradiction
  - Since only one assumption was made, it must be wrong (i.e., not false after all, but rather true)

To Prove **$L \Rightarrow \neg Q$**

Step1:	$\neg (L \Rightarrow \neg Q)$	Assume negation
Step2:	$\neg (\neg L \vee \neg Q)$	Rewriting
Step3:	$L \wedge Q$	De morgan’s law
Step4:	L	Simplifying
Step5:	$\neg(S \vee R)$	$L \Leftrightarrow \neg(S \vee R)$

$\neg(S \vee R) \Rightarrow Q$ Simplifying

$\Rightarrow S \vee R$ This is a contradiction.

The previous step says $\neg$(**S** $\vee$ **R)** and later it become (**S** $\vee$ **R)**

Thus, Step1 is incorrect. Therefore, $L \Rightarrow \neg Q$

To Prove $L \Rightarrow \neg Q$ (Alternative solution)

Step1:	$Q \Rightarrow (S \vee R)$	start with this formalism
Step2:	$\neg Q \vee (S \vee R)$	rewriting
Step3:	$\neg Q \vee \neg L$	using $L \Leftrightarrow \neg(S \vee R)$
Step4:	$\neg L \vee \neg Q$	commutative
Step5:	$L \Rightarrow \neg Q$	rewriting

Therefore, $L \Rightarrow \neg Q$

Specification Types

- Interface Specification (how subsystems are going to interact with each other but not about the details of a particular subsystem by itself)

- Behavioral specification (concentrate on the behavior of individual subsystem or individual module but does not concern with the internals of the subsystem)

Interface Specification

- Large systems are decomposed into subsystems with well-defined interfaces between these subsystems (how a module is built or how it does none of the concerns)
  - It deals with Operations
    - return type
    - parameters
    - name
  - Module contains different operations.
  - Sum of all the operational specification wrote out is the Interface specification.

Interface Specification (Cont..)

- Specification of subsystem interfaces allows independent development of the different subsystems.
- Interfaces may be defined as abstract data types or object classes.
- The algebraic approach to formal specification is particularly well-suited to interface specification
  - interface does not concern a state.

Behavioral specification

- Algebraic specification can be cumbersome when the object operations are not independent of the object state
- Model-based specification exposes the system state and defines the operations in terms of changes to that state
- The Z notation is a mature technique for model-based specification. It combines formal and informal description and uses graphical highlighting when presenting specifications

Key points

- Formal system specification complements informal specification techniques
- Formal specification are precise and unambiguous. They remove areas of doubt in a specification.
- Formal specification forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system.