Software Development Life Cycle (SDLC)

Challenges for Software Engineers

- **Why does it take so long to get software finished?**
- **Why are development costs so high?**
- **Why can't we find all errors before we give the software to our customers?**

- **An Systematic approach for managing SDLC will help**

SDLC

Software Life Cycle

- **A software life cycle is the series of identifiable stages that a software product undergoes during its lifetime**
- **The first stage is called scope**
- **The subsequent stages are: requirement Mgmt: analysis and specification, design, coding, testing, and maintenance**
- **Each of these stages is called a *life cycle phase***

Phases of Software Development

1. Project Startup - Concept, Proposal, Scope
2. Requirements and Analysis.
3. High level Design.
4. Low level Design.
5. Construction - Unit test, Code inspection.
6. Integration and System tests.
7. Replication, delivery, installation.
8. Acceptance Testing (Requirements mapped)
9. Training, Documentation
10. Project windup - Checklist, Report, Lessons learnt.
11. Maintenance (may involve all the above)

Benefits of SDLC

- Increasing quality
- Reducing project cost and schedule
- Improving manageability

SDLC Process Models

Waterfall Model

Linear Process Models

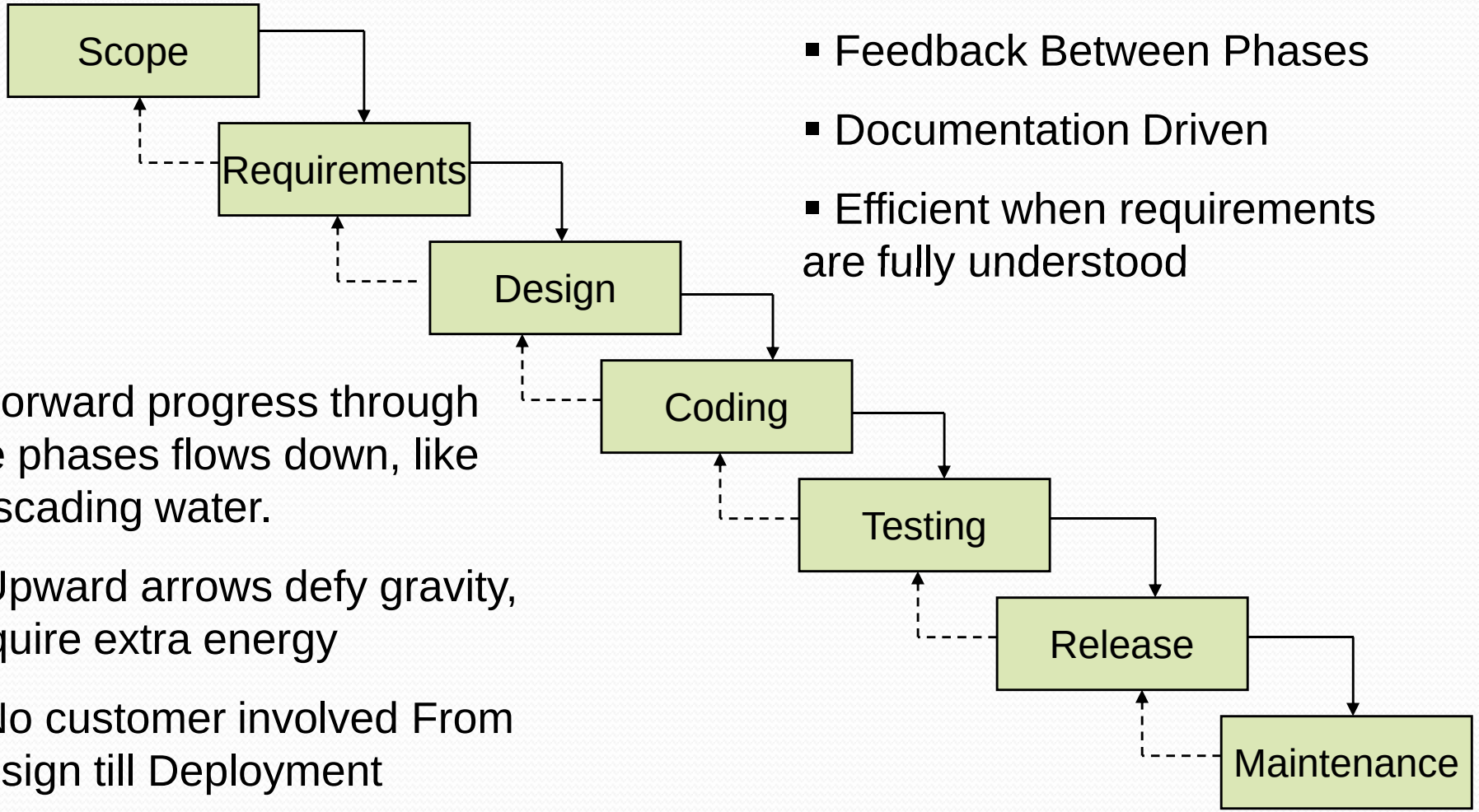
(Contd…)

Waterfall Model

- **Pre-requisites/Pre-conditions**
  - **Requirements are all available**
  - **Requirements do not change**
  - **Relationship with customer is mature (means we understand their needs/requirements properly and they also have confidence in us)**

Linear Process Models

Waterfall Model



- Sequential Steps (Phases)
- Feedback Between Phases
- Documentation Driven
- Efficient when requirements are fully understood

- Forward progress through the phases flows down, like cascading water.
- Upward arrows defy gravity, require extra energy
- No customer involved From Design till Deployment

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule ##

Waterfall Deficiencies

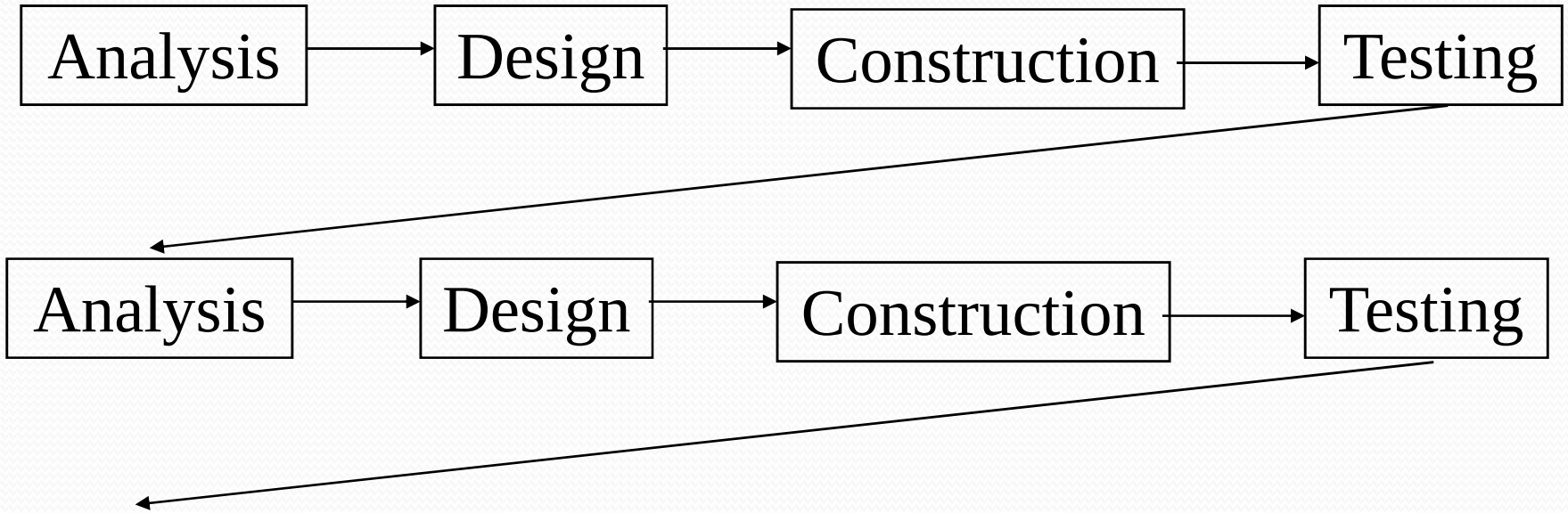
- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (High Risk of wrong product) ##

When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform. ##

Linear Process Models

Incremental Model



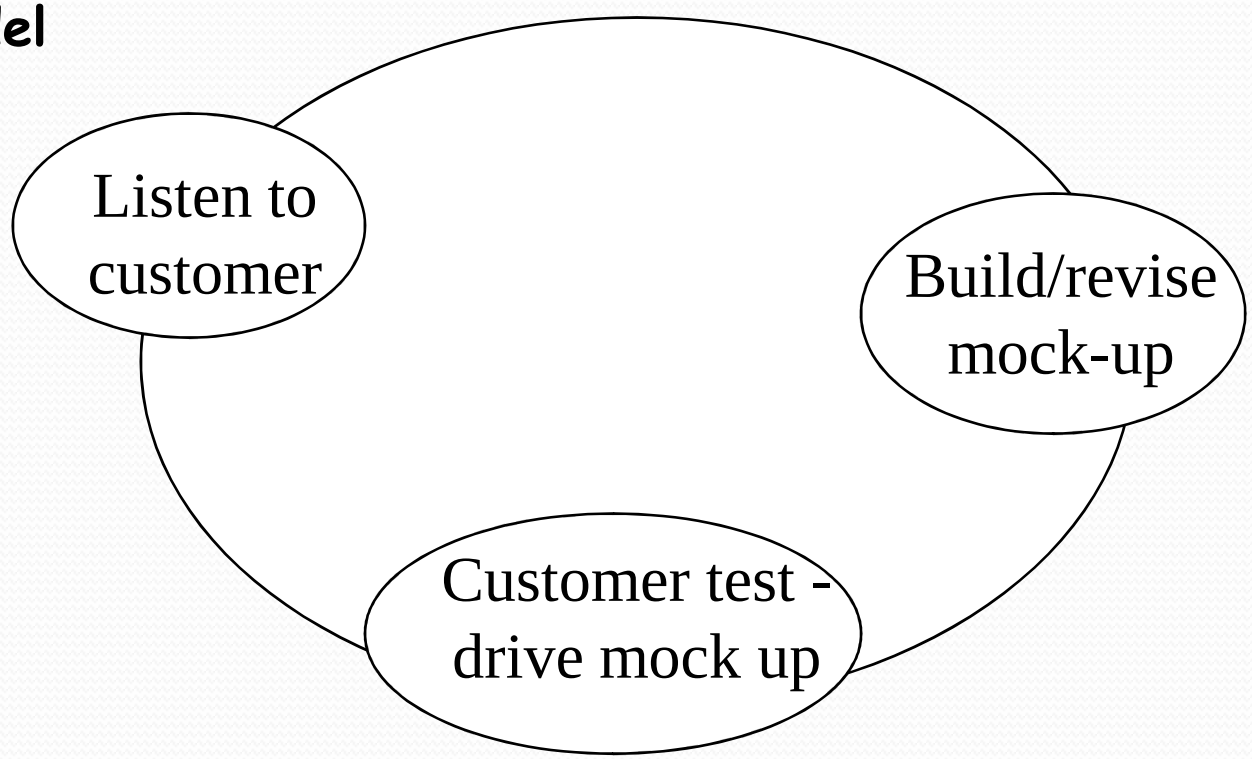
- Adding requirements one by one.
- When,

  All the requirements are not available (Manufac. Comp.)and/or
  Part of the software is needed earlier (SBI-ICICI) and/or
  When the relationship with customer is not mature and/or
  Not enough resources are available

Linear Process Models

(Contd…)

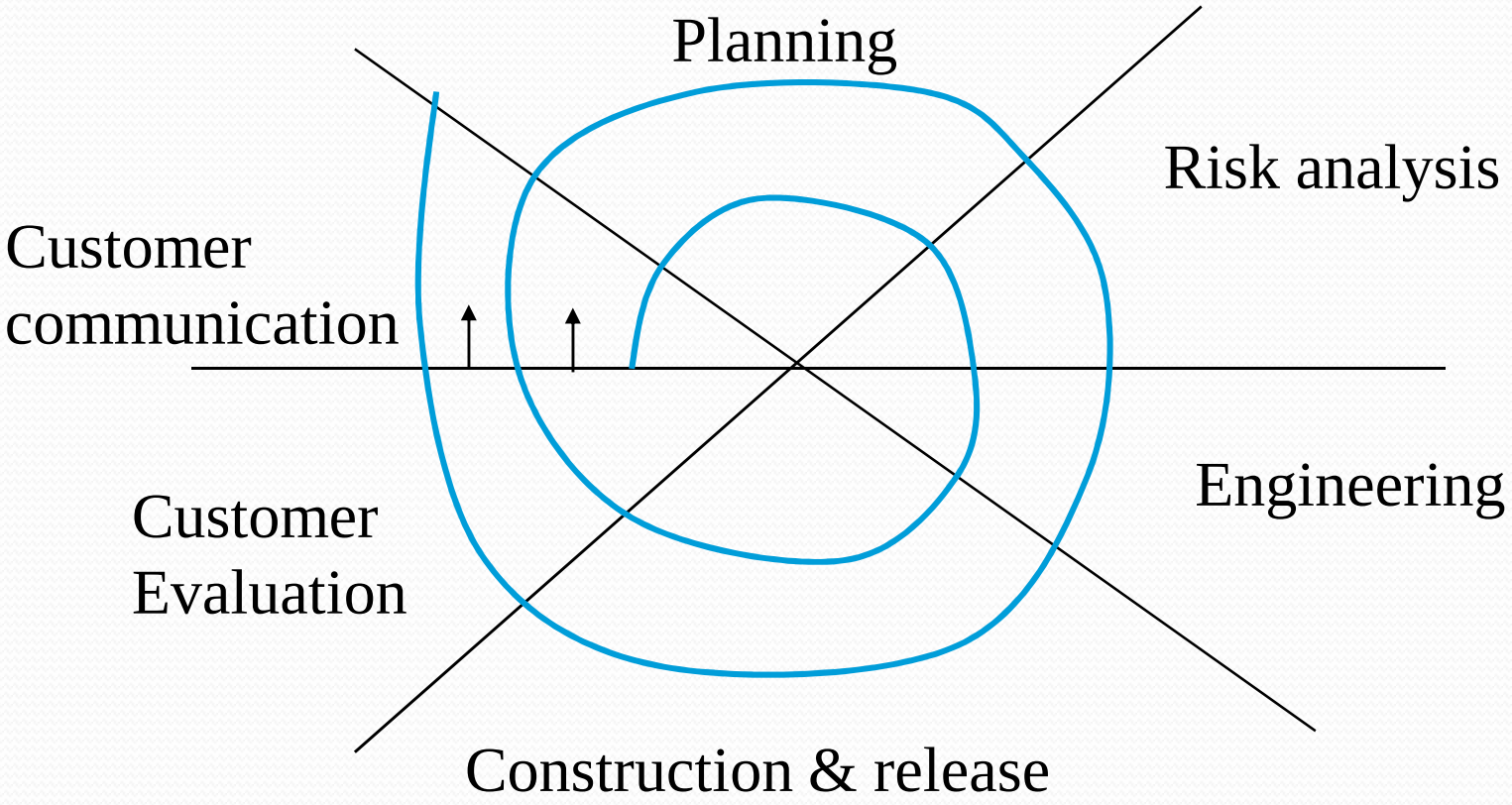
Prototyping Model



- **Customer sticks to prototype for a working software**
- **Developer implements only min. necessary stuff**
- **Build an example system to help elicit requirements**
- **Perfection of Prototype can take too much time**

Evolutionary Process Models (Contd…)

Spiral Model



Evolutionary Process Models (Contd…)

Spiral Model

- Suitable for Software or Products that *evolve* over a period of time
- Uses prototyping as a risk reduction mechanism
- Relies on expertise for success (especially risk assessment)

Incremental vs. Iterative

(Contd…)

- These *sound* similar, and sometimes are equated.
- Subtle difference:
  - **Incremental**: ***add to*** the product at each phase
  - **Iterative**: ***re-do*** the product at each phase
- Some of the models could be used either way

Incremental vs. Iterative

[Example: building a house]

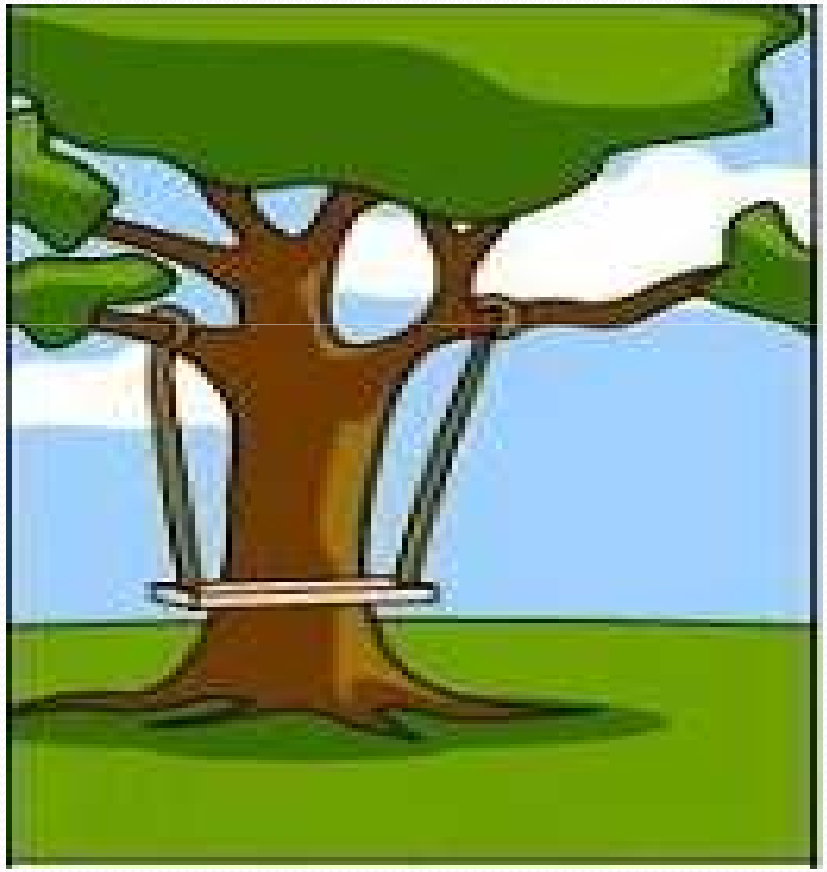
(Contd…)

- **Incremental**: Start with a modest house, keep adding rooms and upgrades to it.
- **Iterative**: On each iteration, the house is re-designed and built anew.
- Big Difference: One can live in the incremental house the entire time! One has to **move** to a new iterative house.

Just for laughs

How the customer explained it…

How the project leader understood it…

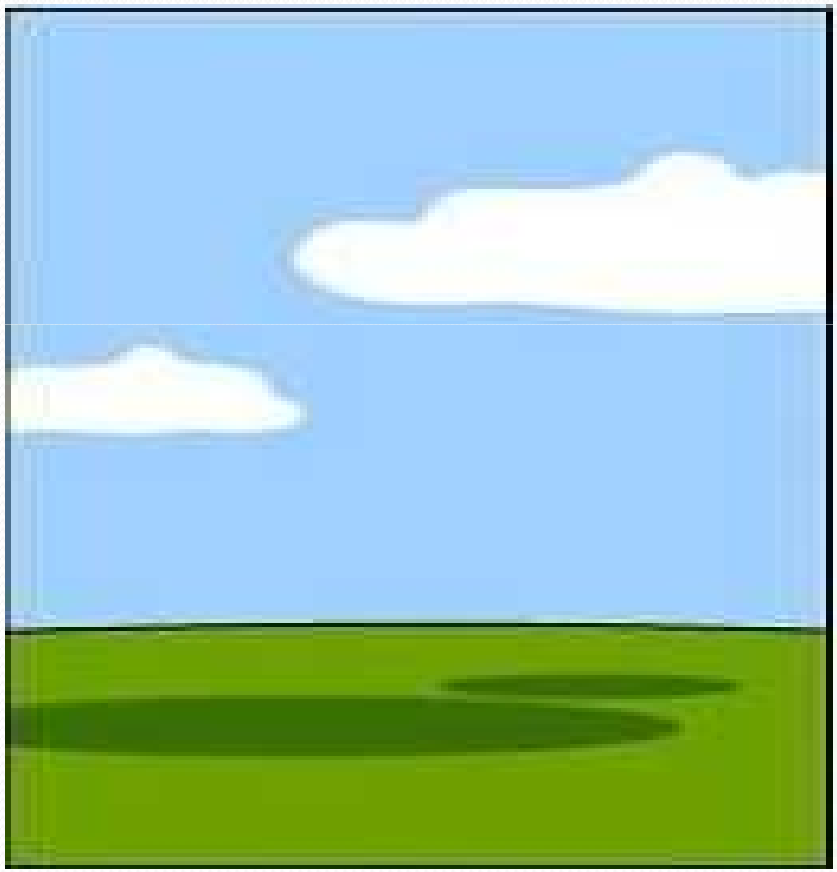


How the system analyst designed it…

How the programmer wrote it…

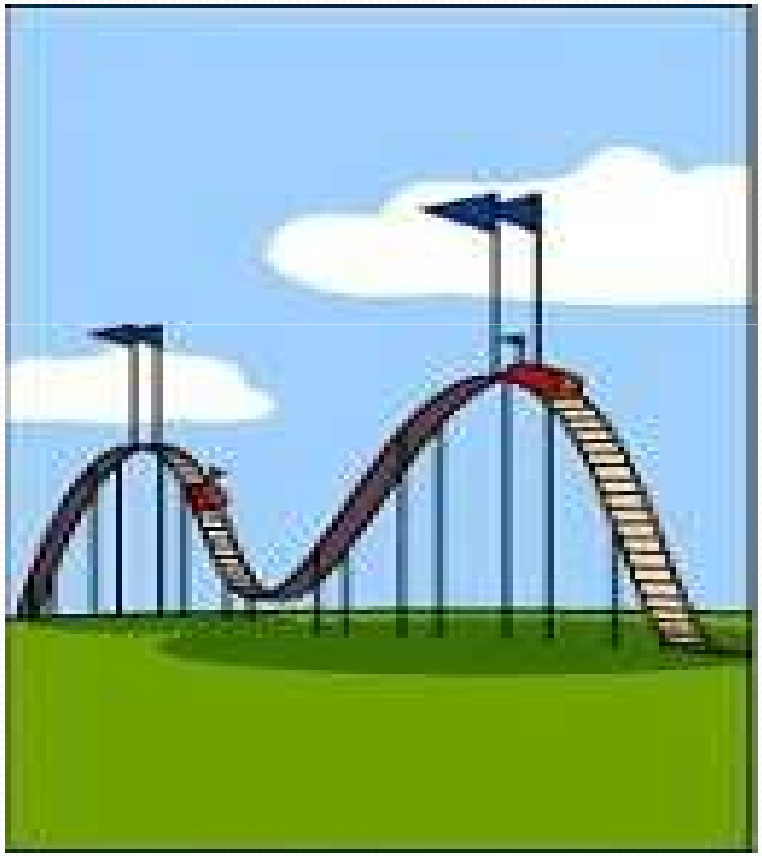
How the Sales Person described it while selling it to the customer…

How the project was documented…

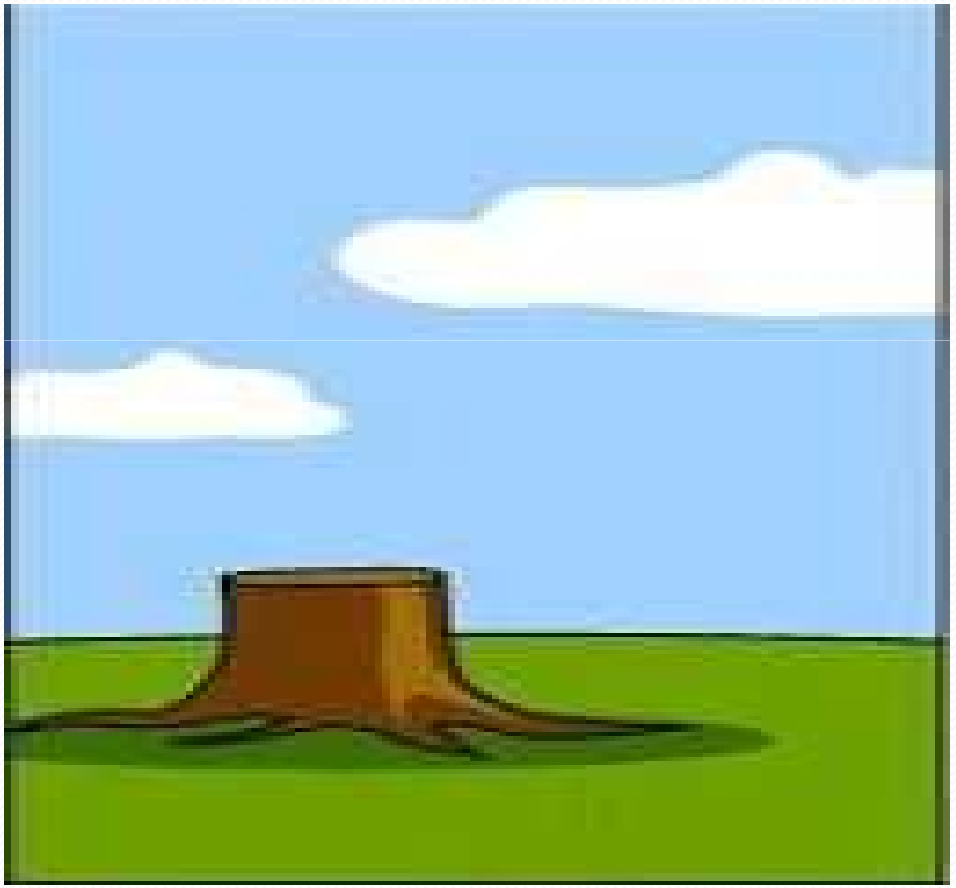


What operations installed…

How the customer was billed…



How it was supported…



What the customer really needed…

End of session