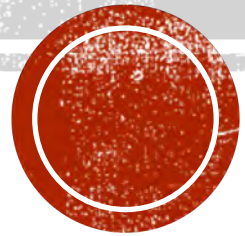
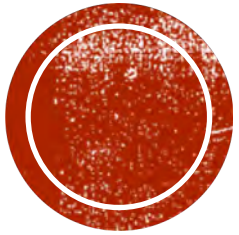
REQUIREMENTS

Software Engineering

Dr. Raj Singh





SOFTWARE ENGINEERING PRINCIPLES

SOFTWARE ENGINEERING KNOWLEDGE

- Technology and tools change frequently.
- New frameworks and effective processes are emerging.
- What you know today might be obsolete in near future.
- The core software engineering principles do not obsolete.
- These core principles are likely to help professionals throughout their career.
- Researchers and professionals find new ways to implement them.

PROCESS GUIDING PRINCIPLES

Be agile	Whether the process model you choose is prescriptive or agile, the basic tenets of agile development should govern your approach.
Focus on quality	Every process activity, action, and task should focus on the quality of the work product that has been produced.
Be ready to adapt	When necessary, adapt your approach to constraints imposed by the problem, the people, and the project itself.
Build an effective team	Build a self-organizing team that has mutual trust and respect.
Effective communication	Projects fail because important information falls into the cracks and/or stakeholders fail to coordinate their efforts to create a successful end product.
Manage change	Mechanisms must be established to manage the way changes are requested, assessed, approved and implemented.
Assess risk	Lots of things can go wrong as software is being developed. It's essential that you establish contingency plans.

COMMUNICATION PRINCIPLES

Listen	Try to focus on the speaker's words, rather than formulating your response to those words.
Prepare	Spend the time to understand the problem before you meet with others.
Stay focused	Keep the conversation moving in a productive direction.
Take notes	Write down all important points and decisions.
Strive for collaboration	Collaboration and consensus occur when the collective knowledge of members of the team is combined.
Move on	Once you agree to something, move on. If you can't agree to something, move on.
Negotiation	It is not a contest or a game. It works best when both parties win.

PLANNING PRINCIPLES

Understand the scope	Scope provides the software team with a destination.
Involve the customer	The customer defines priorities and establishes project constraints.
Changes are inevitable	A project plan is never engraved in stone. As work begins, it very likely that things will change.
Estimate based on what you know	Provide an estimate of effort, cost, and task duration, based on the team's current understanding of the work to be done.
Consider risk	If you have identified risks that have high impact and high probability, contingency planning is necessary.
Be realistic	People don't work 100 percent of every day.
Track and adjust as required	Software projects fall behind schedule one day at a time.

REQUIREMENTS MODELING PRINCIPLES

Understand

The information domain of a problem must be represented and understood.

Define Functions

The functions that the software performs must be defined.

Define Behavior

The behavior of the software must be represented.

Define Boundaries

What is in scope and what is not must be defined.

Analyze

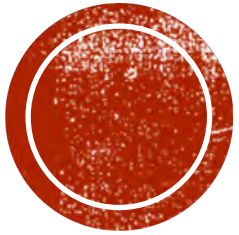
The analysis task should move from essential information toward implementation detail.

DESIGN MODELING PRINCIPLES

- Design should be traceable to the requirements model.
- Always consider the architecture of the system to be built.
- Design of data is as important as design of processing functions.
- Interfaces (both internal and external) must be designed with care.
- Components should be loosely coupled to each other than the environment.
- Design representations (models) should be easily understandable.
- The design should be developed iteratively
- Design should change as requirements change.

IMPLEMENTATION GUIDING PRINCIPLES

Divide and conquer	Stated in a more technical manner, analysis and design should always emphasize separation of concerns (SoC).
Consistency	A familiar context makes software easier to use.
Modularity	Separation of concerns establishes a philosophy for software. Modularity provides a mechanism for realizing the philosophy.
Look for patterns	If there is a recurring problem, there is a pattern and ways to solve it.
Reusability	Build reusable components. Reusability expedites development.
Testing	Test everything you implement. Good testing drives good quality.



UNDERSTANDING REQUIREMENTS

REQUIREMENTS ENGINEERING

Inception	ask a set of questions
Elicitation	elicit requirements from all stakeholders
Elaboration	create an analysis model that identifies data, function and behavioral requirements
Negotiation	agree on a deliverable system that is realistic for developers and customers
Specification	Requirements specification document
Validation	a requirements review mechanism
Requirements Monitoring	Manage & update as required

INCEPTION

Identify stakeholders

"who else do you think I should talk to?"

Recognize multiple points of view

Work toward collaboration

Prepare set of questions

Who is requesting?

Who will use the solution?

How success will be measured?

ELICITING REQUIREMENTS

Meetings by both software engineers and customers

Rules for preparation and participation are established

An agenda is suggested

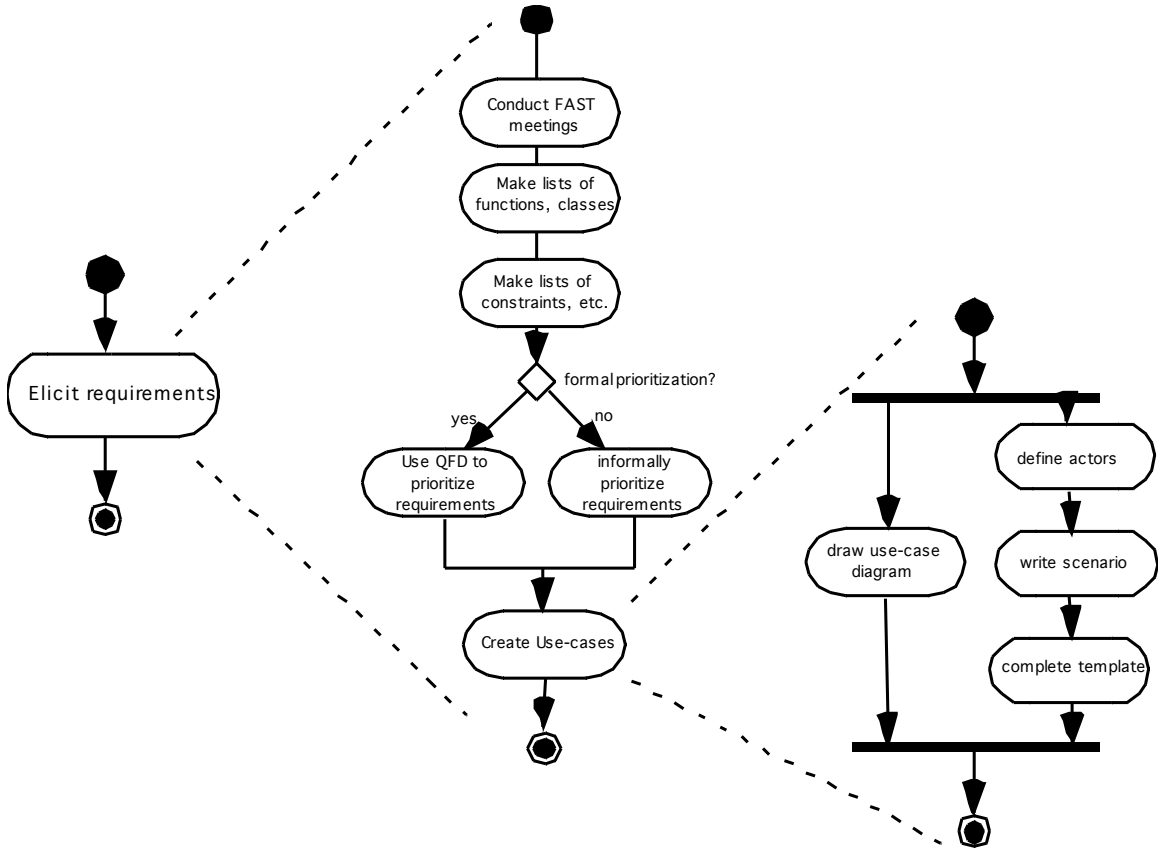
A "facilitator" controls the meeting

The goal

Identify the problem

Propose the solution

ELICITING REQUIREMENTS



ELABORATION

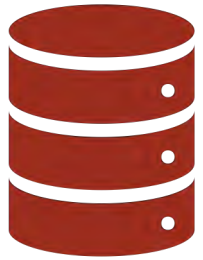
- A statement of need and feasibility
- Scope for the system or product
- List of customers, users, and other stakeholders
- Description of the system's technical environment
- A list of requirements and the domain constraints that apply to each
- A set of usage scenarios that provide insight into the use of the system or product
- Any prototypes to better define requirements.

FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS

Functional

Directly relates to requirements document.

What is being asked and what needs to be built?



Non-Functional

Quality, performance, security, or general system constraint.

Non-spoken requirements. All software systems must have these.

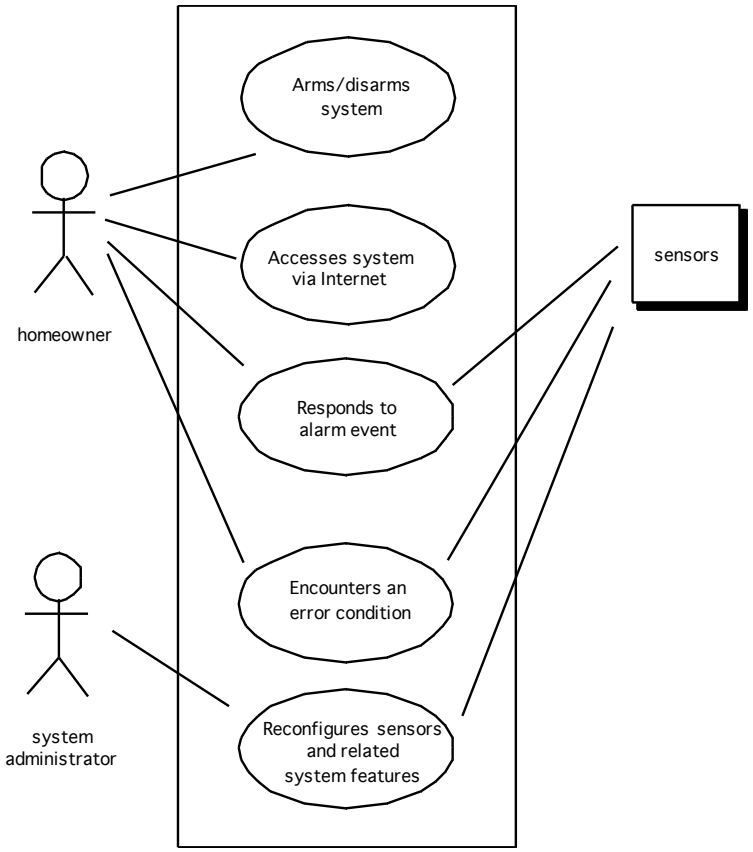
BUILDING THE ANALYSIS MODEL

Scenario-based model	Functional Use-case
Class-based model	Implied by scenarios
Behavioral model	State diagram
Flow-oriented model	Data flow diagram

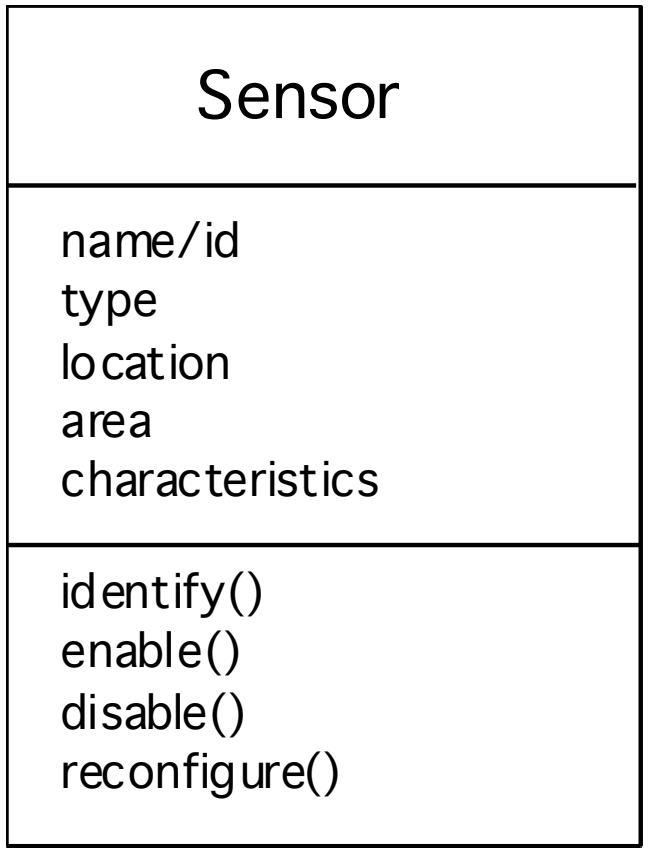
USE-CASES

- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"
  - a person, device, or a system that interacts with the software in some way
- Who is the primary actor, the secondary actor (s)?
- What are the pre and post conditions?
- What main tasks or functions are performed?
- Are there any dependencies?
- How actor will interact with the system?
- What information will the actor acquire, produce, or change?

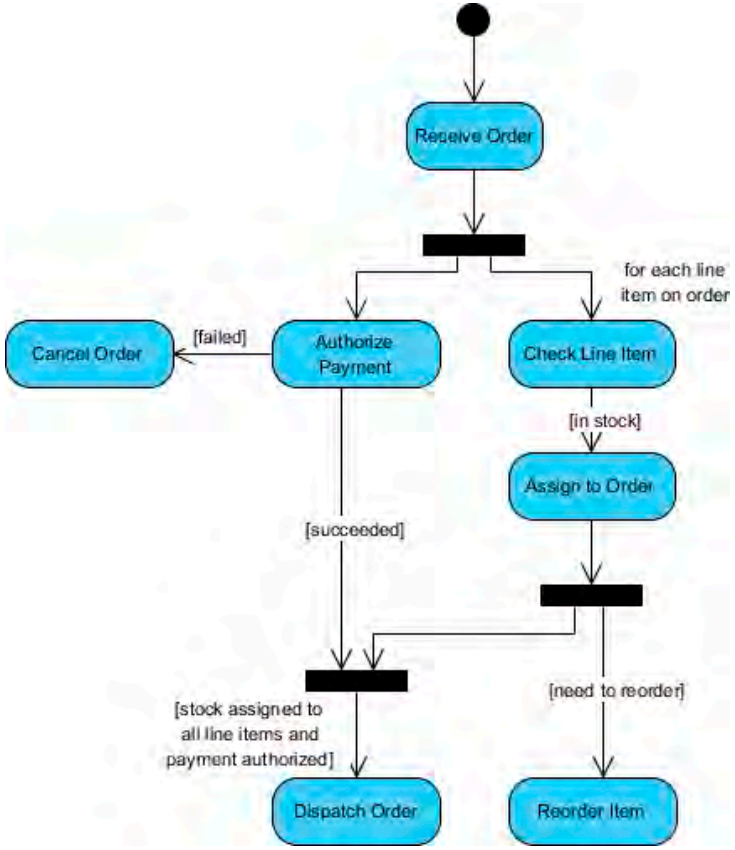
USE-CASE DIAGRAM



CLASS DIAGRAM



STATE DIAGRAM



NEGOTIATING REQUIREMENTS

Identify the key stakeholders

These are the people who will be involved in the negotiation

Determine each of the stakeholders "win conditions"

Win conditions are not always obvious

Negotiate

Work toward a set of requirements that lead to "win-win"

VALIDATING REQUIREMENTS

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Is each requirement achievable?
- Does the requirements model properly reflect the information, function and behavior of the system to be built?
- Have requirements patterns been used to simplify the requirements model?

REQUIREMENTS MONITORING

Distributed debugging	uncovers errors and determines their cause
Run-time verification	determines whether software matches its specification
Run-time validation	assesses whether evolving software meets user goals
Business activity monitoring	evaluates whether a system satisfies business goals
Evolution and codesign	provides information to stakeholders as the system evolves

REFERENCE

- Roger Pressman, Software Engineering: A Practitioner's Approach, 8th edition, McGraw Hill, ISBN 0078022126