
Chapter 4 — Developing Requirements
Domain Analysis #81C784
What is domain analysis?
Domain expert
Domain document (Intro, Glossary, Customers, Environment, Tasks, Competing SW)
Benefits (Faster dev, Better system, Anticipate extensions)
Starting Point for Projects #64B5F6
Greenfield projects
Defining Problem & Scope #FFD54F
Problem statement (short & succinct)
Scope techniques
List possibilities
Exclude overly-broad items
Narrow / widen as needed
Example: University registration system
What is a Requirement? #FFB74D
Functional vs Constraint
Requirements document = negotiated stakeholder agreement
Types of Requirements #4DB6AC
Functional requirements
Inputs, Outputs, Data storage, Computations, Timing/sync
Quality requirements
Verifiable: response time, throughput, reliability, availability, recovery, maintainability, reusability
Platform requirements
Process requirements
Use-Cases (how users use system) #BA68C8
Definition: sequence of user actions to achieve a task
Use case model = set of use cases + relations
Properties of a good use case
Full sequence, user interaction only, UI-independent, verifiable steps
Scenarios = instance of a use case
How to describe a use case
Name, Actors, Goals, Preconditions, Summary, Related UC, Steps (2-col), Postconditions
Relations: Extension, Inclusion, Generalization
Benefits of use-case-driven dev (scope, planning, tests, manuals)
Limitations (must validate; not everything covered)
Use-Case Examples (Open File family) #90A4AE
Open file (basic)
Open by typing name (specialization)
Open by browsing (specialization; includes Browse for file)
Attempt to open non-existent file (extension)
Browse for file (inclusion)
Gathering & Analyzing Requirements #A1887F
Observation (documents, shadowing, videotaping)
Interviewing (series, future vision, alternatives)
Brainstorming (moderator, trigger question)
JAD (intensive brainstorming variant)
Prototyping (paper, UI mockups, rapid prototypes)
Use-case analysis (identify actors, tasks)
Types & Structure of Requirements Documents #B0BEC5
Hierarchy for large systems (definition, specification, subsystem, sub-subsystem)
Two extremes: informal outline vs exhaustive spec
Level-of-detail factors: system size, interfaces, readership, stage, domain/tech experience, cost of faults
Reviewing Requirements #E57373
Each requirement must:
Have benefit > cost
Be important to problem
Be clear, unambiguous, consistent, verifiable, realistic, uniquely identifiable
Not over-constrain design
Document should be complete, organized, agreed; traceability
Requirements Document Contents (recommended) #AED581
Problem, Background, Environment & models, Functional reqs, Non-functional reqs
Managing Changing Requirements #FF8A65
Causes (business, tech, improved understanding)
Requirements analysis never stops
Assess benefit vs cost for changes
Small UI changes cheap; large changes require careful assessment
Avoid making system bigger unnecessarily
Difficulties & Risks in Analysis #EF9A9A
Lack of domain understanding — mitigate: domain analysis, prototyping
Rapidly changing reqs — mitigate: incremental dev, flexibility, reviews
Attempting too much — mitigate: document boundaries, estimate carefully
Conflicting requirements — mitigate: brainstorming, JAD, competing prototypes
Hard to state precisely — mitigate: break into simple sentences, prototypes
Closing / Notes #B39DDB
Traceability, verification, stakeholder agreement
Use cases for planning, testing, validation