Requirements Analysis

Analysis and Design?

- **Analysis** emphasizes an investigation of the problem and requirements, rather than a solution.
  - Analysis = requirements analysis + object analysis.
- **Requirement analysis** is a prerequisite activity.
  - Who will use the system?
  - How will it be used?
  - What are the expectations?
- **Object analysis** is about finding domain objects and concepts.

- **Design** is a conceptual solution that fulfills the requirements.
  - E.g. a description of software objects, and databases schema.
  - Software objects are inspired by domain objects, but not necessary the same.

Key UP Idea: **Iterative** Development

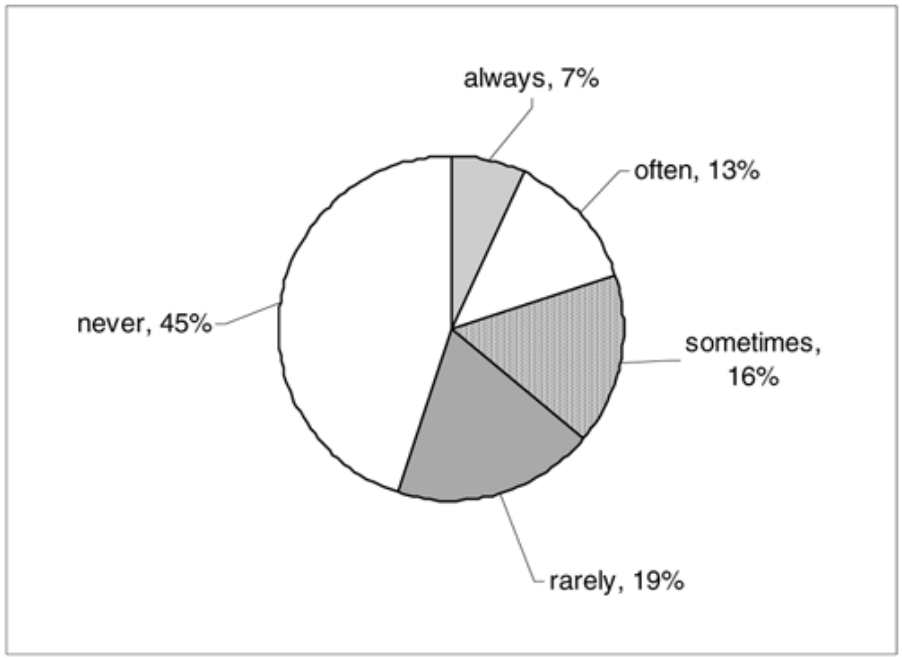
- Iterative development: series of short, fixed-length mini-projects called **iterations**.

- The outcome of each iteration is a **tested**, **integrated**, and **executable** system.

- Each iteration includes its own requirement analysis, design, implementation, and testing activities.

Why Waterfall is bad?

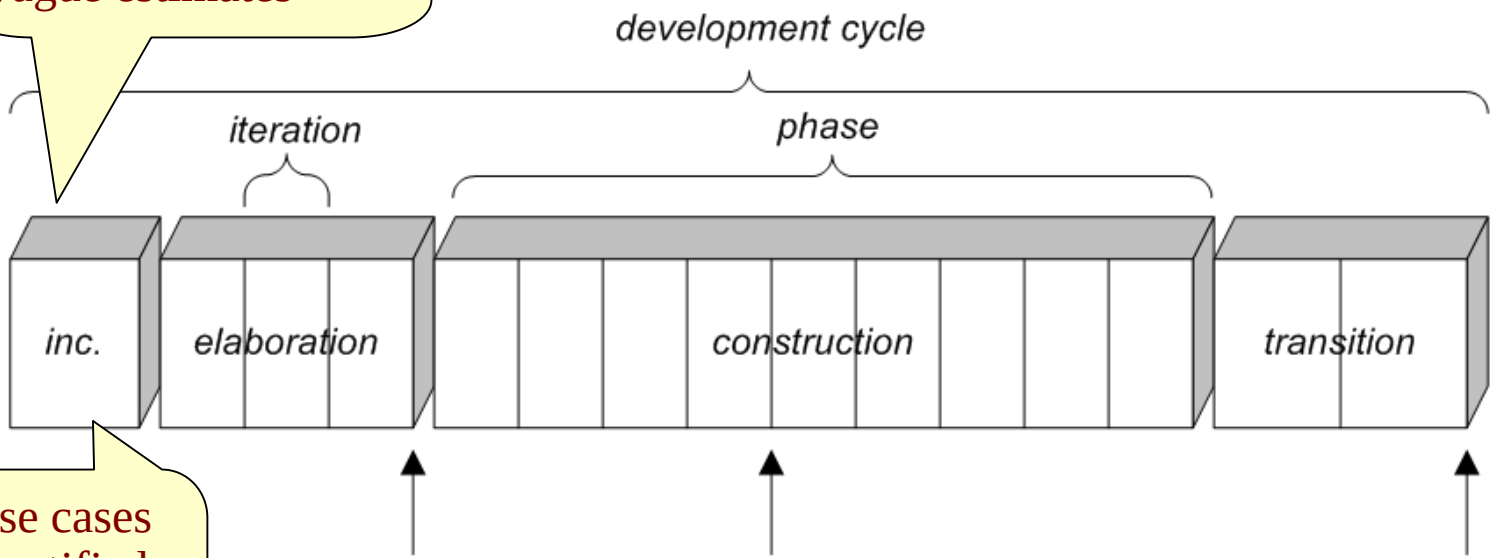
- Statistically, 35%-50% of requirements change in a software project.
- Any method (such as Waterfall) that attempts to freeze requirements at the start is fundamentally flawed.
- Also many of the prematurely early specified features are not useful in the final software product?



UP Phases

Goal of Inception:
Feasible?
Buy and/or build?
Should we proceed or stop?

approximate vision
business case
vague estimates



Use cases identified.
10% only written up.

milestone

An iteration end-point when some significant decision or evaluation occurs.

release

A stable executable subset of the final product. The end of each iteration is a minor release.

increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is released for production use.

Iterative Analysis and Design

1. Before iteration-1, hold the first timeboxed requirements workshop (2 days). Business and development people are present.
   - On the morning of day one, identify the names of the use cases. The analysis will not be perfect.
   - Ask the chief architect and business people to pick 10% from this high-level list (such as 10% of the 30 use case names), which have a blending of three qualities:
     1) architecturally significant
     2) high business value (features business really cares about)
     3) high risk (such as "be able to handle 500 concurrent transactions").

     Perhaps three use cases are identified: UC2, UC11, UC14.
   - For the remaining 1.5 days, do intensive detailed analysis for these three use cases (write them up).

Iterative Analysis and Design

2. Before iteration-1, hold an iteration planning meeting

   a subset from UC2, UC11, and UC14 are chosen to **design**, **build**, and **test** within a specified time (for example, four-week timeboxed iteration).

3. Do iteration-1 over four weeks.

   – On the first two days, developers and others do **modeling** and **design** work in pairs, sketching UML diagrams at many whiteboards in a **common war room**…

   – Then the developers take off their "modeling hats" and put on their "programming hats."

   – Much **testing** occurs.

   – One week before the end, ask the team if the original iteration goals can be met; if not, de-scope the iteration, putting secondary goals back on the "to do" list.

   – On Tuesday of the last week there's a **code freeze**; all code must be **checked in**, **integrated**, and **tested** to create the **iteration baseline**.

   – On Wednesday morning, **demo** the partial system to external **stakeholders**, to show **early visible progress**. **Feedback** is requested.

   – Thursday & Friday prepare for the next iteration.

Artifacts started in Inception

- Vision and business case
- Use-Case model
- Supplementary Specs (everything not in UC's)
- Glossary
- Risk list

FURPS model for Requirements

- In UP requirements are organized according to the FURPS model.
  - **Functional (or Behavioral)**

    features, capabilities, security
  - **Usability**

    human factors, help, documentation
  - **Reliability**

    frequency of failure, recoverability, predictability
  - **Performance**

    response times, throughput, accuracy, availability, resources
  - **Supportability**

    Adaptability, maintainability, internationalization, configurability

Recorded in **Use-Cases**

Other four are called **quality requirements**

Use-Cases – **Goals** and **Stories**

- Customers have **goals** and want computer systems to help meet them.
  - E.g. recording sales, updating inventory etc.

- Use-cases are **stories** of using the system to meet goals.

- Use-cases can be:
  - **Brief –** one paragraph summary of the main success scenario
  - **Casual –** multiple informal paragraphs covering various scenarios
  - **Fully dressed –** the most elaborate, where all steps and variations are written in detail

A brief format use-case

Process Sale:

A customer arrives at a checkout with items to purchase.

The cashier uses the POS system to record each purchased item.

The system presents a running total and line-item details.

The customer enters payment information, which the system validates and records.

The system updates inventory.

The customer receives a receipt from the system and then leaves with the items.

- What about other formats?

  (first some definitions)

First Some Definitions

- **Actor** is something with behavior. E.g.
  - **Person** identified by role (cashier for e.g.)
  - **Computer system** (the one for inventory for e.g.)
  - **Organization** (government for e.g.)

- **Scenario** is a sequence of **actions** and **interactions** between actors and system.

- **Use case** is a collection of *related* success and failure scenarios.
  - **Contract** of how the system will behave.

Text Docs and Black-Box Nature

- Use cases are **text docs**, not diagrams, and use case modeling is primarily about writing text, not drawing.
- However, UML defines a **use-case diagram** to illustrate the names of use cases and actors, and their relationships.
  - **Context diagram** of a system and its environment.
- Use cases are of **black-box** nature; they describe what the system should do, and not how to do it.
- **Key idea:** to reduce the importance or use of detailed old-style feature lists.

Fully dressed example: Process Sale

Use Case UC1: Process Sale

Start with a verb.

Primary Actor: Cashier

Stakeholders and Interests:

- Cashier:
  - Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.
- Customer:
  - Wants purchase and fast service with minimal effort.
  - Wants proof of purchase to support returns.

- Company:
  - Wants to accurately record transactions and satisfy customer interests.
  - Wants to ensure that Payment Authorization Service transact. are recorded.
  - Wants automatic and fast update of accounting and inventory.
  - Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable.
- Government Tax Agencies:
  - Want to collect tax from every sale. May be multiple agencies, such as national and provincial.
- Payment Authorization Service:
  - Wants to receive digital authorization requests in the correct format and protocol.
  - Wants to accurately account for their payables to the store.

Preconditions:

Cashier is identified and authenticated.

Postconditions:

Sale is saved.

Receipt is generated.

Accounting and Inventory are updated.

Tax is correctly calculated.

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Alternative Flows:

Condition

Handling

*a. At any time, System fails:

Ensure transaction state can be recovered from any scenario step.

1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.

2a. System detects anomalies preventing recovery:

1. System signals error to Cashier, records error, and enters a clean state.
2. Cashier starts a new sale.

'*' means, it can happen at any step of main scenario

3a. Invalid identifier:

1. System signals error and rejects entry.

Observe numbering:
First number corresponds
to a step in main scenario

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):

1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:

1. Cashier cancels sale on System.

4a. System generated item price is not wanted (e.g., Customer complained about something and is offered a lower price):

1. Cashier enters override price.
2. System presents new price.

5a. System detects failure to communicate with external tax calculation system service:

1. System restarts service on the POS node, and continues.

1a. System detects that the service does not restart.

1. System signals error.

2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):

1. Cashier signals discount request.

2. Cashier enters Customer identification.

3. System presents discount total, based on discount rules.

6a. Customer says they intended to pay by cash but don't have enough cash:

1. Customer tells Cashier to cancel sale.
2. Cashier cancels sale on System.

7a. Paying by cash:

1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records cash payment.

7b. Paying by credit card:

…

7c. Paying by debit card:

…

Too long.
Good candidate for a
separate, more detailed UC.

Special Requirements:

– Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.

– Credit authorization response within 30 seconds 90% of the time.

– Pluggable business rules to be insertable at steps 3 and 7.

– . . .

Technology and Data Variations List:

3a. Item identifier entered by bar code laser scanner or keyboard.

3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.

7a. Credit account information entered by card reader or keyboard.

7b. Credit payment signature captured on paper receipt. But within two years, we predict many customers will want digital signature capture.

Alternate flows guideline

- What's the difference?
  - 5a. System detects failure to communicate with external tax calculation system service
  - 5a. External tax calculation system not working

- Former style preferred because this is something the system can **detect**;
- The latter is an **inference**.

UC Guideline: UI-free style

Contrasting Examples

- **Essential Style**
  1. Administrator identifies self.
  2. System authenticates identity.
  3. …
- **Concrete Style—Avoid During Early Requirements Work**
  1. Adminstrator enters ID and password in dialog box (see Picture 3).
  2. System authenticates Administrator.
  3. System displays the "edit users" window (see Picture 4).
  4. …

UC Guideline: Write Terse UC's

- Do you like to read lots of requirements?
- Probably not!

- So, write terse use cases.
- Delete "noise" words.
  - "System authenticates…" rather than
  - "The System authenticates…"

How should use cases be discovered?

- Which of these is a valid use case:
  1. Negotiate a supplier contract
  2. Handle returns
  3. Log in

- An argument can be made that all of these are use cases at different levels, depending on the system boundary, actors, and goals.
  - A more practical question is:
  - "What's a useful level to express use cases for **application requirements analysis**?"

Tests

Boss Test

- Your boss asks, "What have you been doing all day?"
- You reply: "Logging in!" Is your boss happy?
- If not, the UC fails the Boss Test.

Elementary Business Process (EBP) Test

- A use-case should describe:

> *A task performed by **one person** in **one place** at **one time**, which adds measurable business value.*

So…

- Which of these is a valid use case:
  1. Negotiate a supplier contract
  2. Handle returns
  3. Log in

(1) is too big

(3) is too small

(2) is good to be described with a use case.

In a conversation with the user…

- We want to discover the user goals in a **goal hierarchy**. E.g.
  - **System analyst**: “What are your goals?”
  - **Cashier**: “One, to quickly log in. Also, to capture sales.”
  - **System analyst**: “What do you think is the higher level goal motivating logging in?”
  - **Cashier**: “I’m trying to identify myself to the system, so it can validate that I’m allowed to use the system for sales and other tasks.”
  - **System analyst**: “Higher than that?”
  - **Cashier**: “To prevent theft, data corruption, and display of private company information.”

Which one?

1. Prevent theft…
2. Identify myself and be validated
3. Capture a sale

Answer: (3)

Use Case Diagrams