Requirement Engineering

Requirements define what the system must do and how well. Over two weeks we discover, write, prioritise, validate and manage the requirements of ITC Club Hub.

ISO/IEC/IEEE 29148:2018 | User stories · Gherkin · MoSCoW | SRS · traceability · change control | Two weeks · Lab-05 in two parts

AGENDA

What we will cover

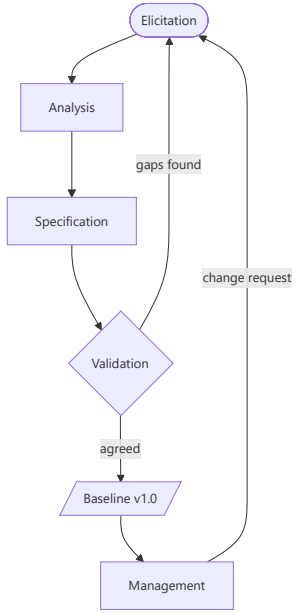
1. Functional and non-functional requirements; user and system requirements
2. Stakeholders and their viewpoints
3. Elicitation techniques: interviews, questionnaires, observation, workshops, prototyping
4. User stories with acceptance criteria (INVEST, Given/When/Then)
5. Textual use cases: main flow and alternative flows
6. The software requirements specification and ISO/IEC/IEEE 29148 structure
7. Quality of requirements: unambiguous, complete, consistent, testable
8. Prioritisation: MoSCoW and value versus effort
9. Validation: reviews, prototypes and test cases
10. Requirements management: traceability and change
11. Chapter Quiz
12. Lab-05

Click any item to jump directly to that topic.

Requirements engineering: deciding what to build before building it

- A **requirement** is a statement of something the system must do or a property it must have, written so that it can be checked (ISO/IEC/IEEE 29148:2018).
- **Requirements engineering (RE)** is the set of activities that discover, document, agree on and maintain those statements. SWEBOK Guide v4.0 treats it as its own knowledge area.
- It is a **loop**, not a phase: validation finds gaps, and every change request restarts the cycle.
- Why it matters: a wrong requirement is cheap to fix on paper and expensive once it is coded, tested and deployed (the cost-of-change curve from Chapter 01).

Week	Activities in this chapter
5	Elicitation and analysis (Lab-05 tasks 1-2)
6	Specification, prioritisation, validation, management (tasks 3-5)

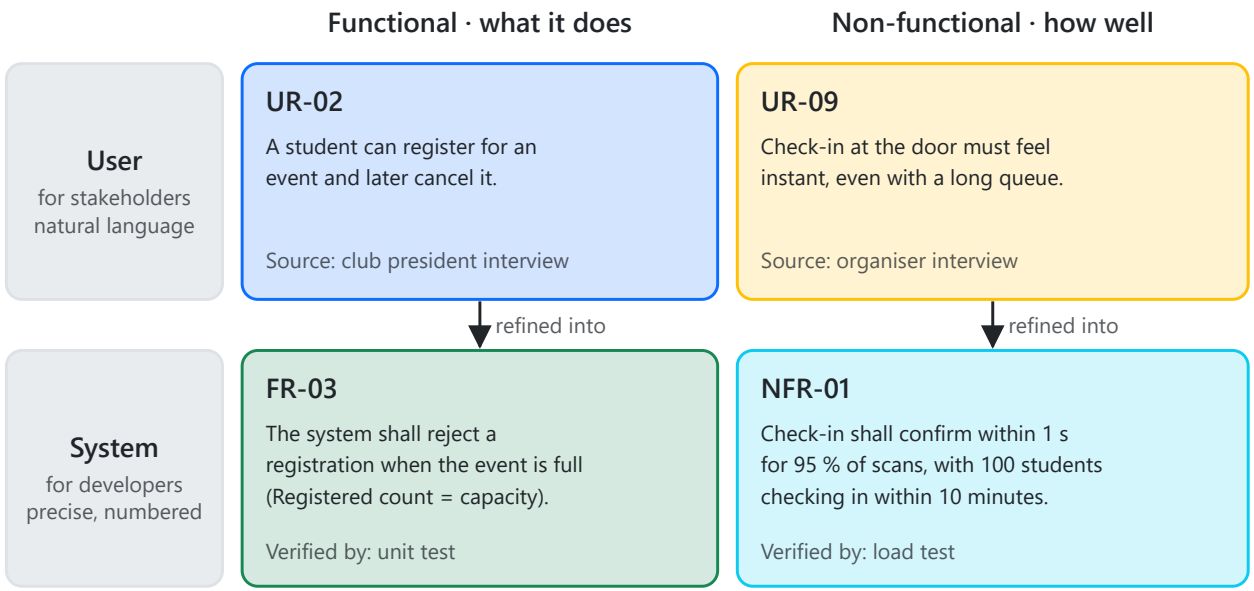


Elicitation discovers needs · analysis classifies them and resolves conflicts · specification writes stories, use cases and the SRS · validation checks them with stakeholders · management keeps them traced and under change control.

Two questions classify every requirement

- **Functional**: a behaviour, what the system does in response to an input (register, cancel, check in).
- **Non-functional**: a quality or constraint, how well it does it (speed, privacy, platforms, language).
- **User requirement**: natural language for clients and users, no internal detail.
- **System requirement**: precise, numbered, detailed enough for developers and testers to agree on "done".

💡 The user / system split follows Sommerville; the two levels are the same need at two levels of detail, not two different needs.



One user requirement becomes several system requirements

```
UR-04  A student can cancel a registration.

FR-06  The system shall reject a cancellation made
       less than 24 hours before the event start.
FR-07  On a valid cancellation the system shall set
       the Registration to Cancelled and free one
       place of the event.
FR-12  If the event has a waiting list, the system
       shall offer the freed place to the first
       Waitlisted student.
NFR-05 The result of a cancellation shall be shown
       to the student within 2 seconds.
```

⚠ Business rules such as "cancellation closes 24 hours before the start" are **functional**: they change what the system does. Do not hide them in a "constraints" list.

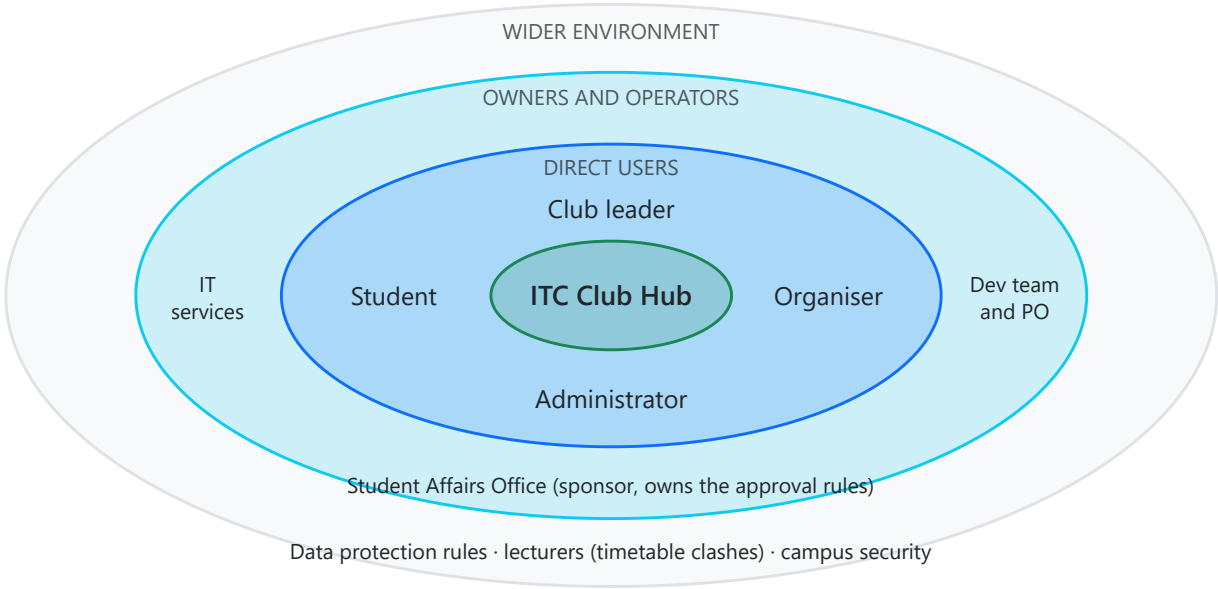
Non-functional category (ISO/IEC 25010:2023)	**Club Hub example**
Performance efficiency	Check-in confirms within 1 s for 95 % of scans
Interaction capability (usability)	A first-time student registers in under 60 s without help
Security	Only an event's organisers can see its attendance list
Reliability	No registration is lost when the program is closed mid-save
Compatibility	The CSV export opens in LibreOffice Calc and Excel
Maintainability	Business rules live in `RegistrationService` and have unit tests
Flexibility (portability)	Builds with GCC, Clang and MSVC from one `CMakeLists.txt`

Constraints are non-functional requirements imposed from outside: C++20, file storage in CSV or JSON, the institute's data protection rules (Chapter 03), the week-14 deadline.

Who has a stake in ITC Club Hub?

- A **stakeholder** is anyone who affects or is affected by the system, including people who never open it.
- The **onion model** places them in rings: direct users, owners and operators, then the wider environment.
- Each stakeholder has a **viewpoint**: the part of the problem they see and the qualities they care about.
- Missing a stakeholder means missing requirements: forget the organiser and nobody asks how check-in works with weak Wi-Fi.

In the labs the instructor or teaching assistant plays the client and can answer as any of these roles; your Product Owner is the team's contact.



Different viewpoints produce conflicting requirements

Stakeholder	Wants	Worries about
Student	Find events fast, register from a phone, a reminder	Personal data shown to other students; missing an event
Club leader (president)	Full rooms, publish events quickly, attendance figures	No-shows, administrative delay
Organiser	A quick check-in, accurate head count, list of walk-ins	Queues at the door, weak Wi-Fi in the hall
Administrator	Only genuine clubs, a monthly report across clubs	Fake clubs, room safety limits
Student Affairs Office	Evidence of club activity for funding decisions	Privacy complaints, reputational risk

⚡ Worked conflict: overbooking

President: "Accept 10 % more registrations than seats, some always stay away."

Administrator: "The room limit is a safety rule."

Underlying interests: full rooms vs. safety.

Decision: capacity is a hard limit (FR-03); a waiting list (FR-12, Should) fills cancelled places. Recorded with both names and the date.

- Negotiate on the *interest* behind each position, not the position.
- Decide against the project goal; escalate to the sponsor if needed.
- Record the decision, the rationale and the affected requirement ids.

Elicitation: several techniques, because each one misses something

Technique	How it works	Strength	Weakness	Club Hub use
Interview	Prepared questions with one stakeholder, 20 to 45 min	Depth, follow-up questions, reasons behind needs	One view at a time; people describe the ideal, not the real	Club president and organiser (Lab-05 task 1)
Questionnaire	Same closed and short open questions to many people	Numbers from many students cheaply	No follow-up; answers only what you thought to ask	"Which device do you use?" to 200 students
Observation	Watch the real work where it happens	Reveals tacit knowledge people do not mention	Time-consuming; being watched changes behaviour	Stand at the door of a real club event during check-in
Workshop	Facilitated session with several stakeholders together	Conflicts surface and get resolved in the room	Loud voices dominate; needs a good facilitator	President, organiser and administrator agree on approval rules
Prototyping	Show a sketch or clickable mock-up and ask users to try it	Users react to something concrete; finds missing steps	Users may take the mock-up for the finished product	Paper screens of the registration flow (Lab-05 challenge)
Document analysis	Read existing forms, sign-up sheets, rules	Cheap; finds data fields and business rules	Documents may be outdated	Paper sign-up sheet and the club approval form

Tacit knowledge is what people know so well they forget to say it (for example "walk-ins are always allowed if seats remain"). Observation and prototypes catch it; interviews alone usually do not. Combine at least two techniques.

Two 20-minute interview scripts

```
# Interview: club president (20 min)
Goal: how events are organised today
## Opening (2 min)
- Thank, explain the purpose, ask to take notes
## Context (5 min)
1. Walk me through the last event you ran.
2. How did students sign up? What went wrong?
3. How many events per month? How many seats?
## Needs (8 min)
4. What do you check before announcing an event?
5. What happens when an event is full?
6. What do you need to know after the event?
7. Which student data do you really need?
## Wrap-up (5 min)
8. What would make you stop using the app?
9. Who else should we talk to?
- Summarise the top 3 needs, agree next contact
```

```
# Interview: event organiser (20 min)
Goal: understand check-in at the door
## Context (6 min)
1. Describe check-in at your last event.
2. How long was the queue? How many helpers?
3. What do you do with walk-ins?
## Needs (8 min)
4. What must you see when a student arrives?
5. What if the Wi-Fi in the hall is down?
6. When do you need the attendance list?
## Probing (4 min)
7. You said "quick": how many seconds is quick?
8. Can you show me the sheet you use today?
## Wrap-up (2 min)
9. Anything I should have asked but did not?
```

⚠ Avoid leading questions ("You would like a waiting list, right?"). Start open ("Tell me about the last time..."), then narrow down, and turn every adjective into a number (question 7).

From interview notes to a list of raw needs

```
Interview notes · organiser, Robotics Club
2026-10-06 14:00 · notes: Dara · asked: Sokha

Q1  "Last workshop: 120 students, two of us at
     the door with a printed list sorted by name.
     The queue took 25 minutes."
Q4  "I need to see the name and whether they are
     registered. Nothing else."
Q5  "The hall Wi-Fi drops when it is full."
Q7  "Quick means: scan, beep, next. A second."
Obs Walk-ins were let in when seats were free;
     their names were written at the bottom.
```

Id	Raw need (stakeholder's words)	Who
N-01	See the events of all clubs in one place	Student
N-02	Registrations stop at the room size	Organiser
N-03	Check-in is "scan, beep, next", about a second	Organiser
N-04	Check-in keeps working when the Wi-Fi drops	Organiser
N-05	Walk-ins can be added at the door	Organiser
N-06	Attendance figures after each event	President
N-07	A reminder the day before	President
N-08	Phone number to call no-shows	Organiser

Raw needs are not yet requirements: they overlap, some conflict (N-08 against the privacy rules of Chapter 03) and some are solutions in disguise. **Analysis** classifies them and resolves conflicts.

A user story: card, conversation, confirmation

- **Card**: one sentence, "As a *role*, I want *capability*, so that *benefit*".
- **Conversation**: the details agreed with the Product Owner, not written on the card.
- **Confirmation**: acceptance criteria that decide when the story is done (Ron Jeffries' three Cs).
- **INVEST** (Bill Wake, 2003) is a checklist for a good story.

💡 The *so that* part is not decoration: it tells the team what the story is worth and helps the Product Owner prioritise.

US-02 · Register for an event — Must · 5 SP

As a student — WHO: a real role

I want to register for an event — WHAT: no UI detail

so that I am sure of a seat — WHY: the value

Confirmation: acceptance criteria

Given the event has 1 free place
When I register
Then my status is Registered and 0 places are free

Card · Conversation · Confirmation

I — Independent
can be built in any order

N — Negotiable
details agreed in conversation

V — Valuable
worth something to a user

E — Estimable
the team can size it

S — Small
fits easily in one sprint

T — Testable
criteria can pass or fail

Given / When / Then for register, cancel and check-in

```
Feature: US-02 Register for an event
  As a student
  I want to register for an event
  So that I am sure of a seat

  Scenario: Place available
    Given event "Arduino 101" has capacity 30
    And 29 students are registered for it
    When student "e20210001" registers
    Then the registration status is Registered
    And the event has 0 free places

  Scenario: Event full
    Given event "Arduino 101" has 30 of 30 registered
    When student "e20210002" registers
    Then the registration is rejected
    And the message is "Event full"

  Scenario: Already registered
    Given student "e20210001" is registered
    When student "e20210001" registers again
    Then the registration is rejected
    And the message is "Already registered"
```

```
Feature: US-04 Cancel a registration
  Scenario: Cancel in time
    Given the event starts on 2026-10-15 at 17:00
    And student "e20210001" is Registered
    When the student cancels on 2026-10-14 at 16:59
    Then the status becomes Cancelled
    And one place is freed

  Scenario: Too late to cancel
    Given the event starts on 2026-10-15 at 17:00
    When the student cancels on 2026-10-14 at 17:01
    Then the cancellation is rejected
    And the message is "Cancellation closed"

Feature: US-07 Check in at the door
  Scenario: Registered student arrives
    Given student "e20210001" is Registered
    When the organiser scans the student card
    Then the status becomes CheckedIn
    And the confirmation appears within 1 second
```

Given sets up the state, **When** is one action, **Then** is an observable result. Concrete values (30 seats, 16:59 vs 17:01) test the boundary of the rule.

Story smells and how to split big stories

Smell	Bad story	Better
Too big (an epic)	As a club leader I want to manage events	Split into publish, edit, cancel an event
No user value	As a developer I want a CSV repository class	A technical task under the story that needs storage
UI in the story	...I want a blue button on the top right	...I want to register in one step (the UI is negotiable)
No benefit	As a student I want a reminder	...so that I do not miss an event I registered for
Two stories in one	...register and get a reminder and see a map	Three stories, prioritised separately
Untestable criterion	Then the page loads fast	Then the confirmation appears within 2 seconds

Splitting patterns

- **By workflow step**: browse, register, cancel, check in.
- **By business rule**: register first, then "reject when full", "reject duplicates", "reject past events" as separate stories when the base story is too big.
- **By role**: the organiser's attendance view and the administrator's monthly report are different stories.
- **Happy path first**: the waiting list (Should) is its own story, US-09, after the Must stories.

☑ Rule of thumb for this course: a story larger than 8 story points is split before it enters a sprint.

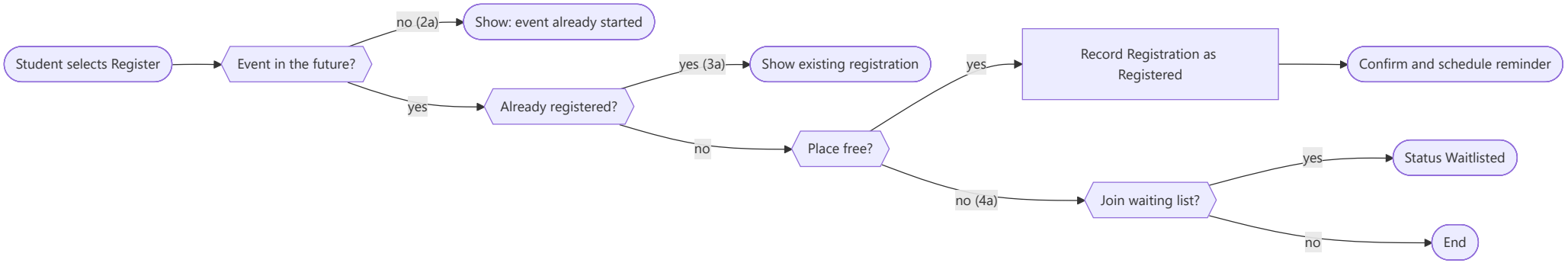
UC-02 Register for an event

```
UC-02  Register for an event
Primary actor: Student            Level: user goal
Stakeholders: Student (wants a seat), Organiser
  (wants an accurate head count)
Precondition: the student is logged in
Trigger: the student opens an event page
Main success scenario:
  1. Student selects Register on the event.
  2. System checks that the event is in the future.
  3. System checks that the student is not registered.
  4. System checks that a place is free.
  5. System records the Registration as Registered
     and reduces the free places by one.
  6. System shows a confirmation and schedules a
     reminder 24 hours before the start.
Alternative flows:
  2a. The event is in the past:
      System shows "Event already started"; ends.
  3a. The student is already registered:
      System shows the existing registration; ends.
  4a. The event is full:
      4a1. System offers the waiting list (Should).
      4a2. Student accepts: status Waitlisted; ends.
      4a3. Student declines: the use case ends.
Postcondition: the student holds at most one
  Registration for the event.
```

- A **use case** describes how an actor reaches one goal with the system, as a numbered dialogue (Cockburn, *Writing Effective Use Cases*, 2001).
- **Actor**: a role outside the system (student, organiser, the clock for reminders).
- **Precondition**: what must be true before; **postcondition**: what is guaranteed after success.
- **Main success scenario**: the happy path, 3 to 9 steps, alternating actor and system.
- **Alternative flows** are numbered after the step where they branch: 4a branches at step 4 when the condition "event is full" holds.
- Each step says *what* happens, never which button or class does it.

Chapter 06 draws the same use case as an oval in a use case diagram; the text is where the behaviour lives.

The flows of UC-02 as a flowchart, and use cases vs. user stories

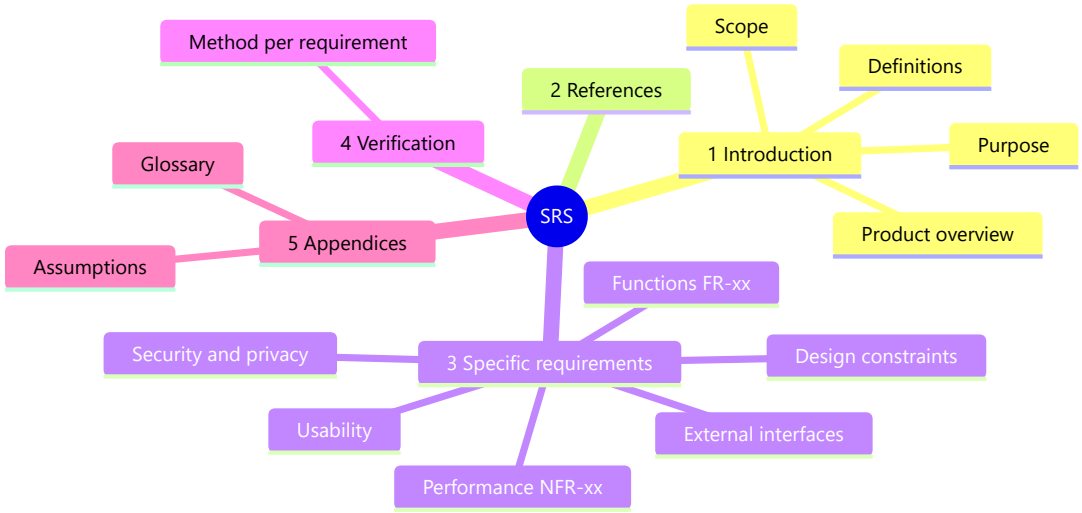


	User story	Textual use case
Size	One sentence + criteria	Half a page to a page
Focus	Value for one role	The full dialogue and its exceptions
Exceptions	One scenario each	Numbered alternative flows
Best for	Backlog, planning, sprints	Complex interactions, many branches

The main success scenario runs from *Student selects Register* to *Confirm and schedule reminder* (answers yes, no, yes); each branch labelled 2a, 3a or 4a is one alternative flow of the text on the previous slide.

They combine well: each alternative flow usually becomes one Gherkin scenario (2a, 3a and 4a match the scenarios of US-02) and later one unit test.

The software requirements specification (SRS)



Outline adapted from the SRS content items of ISO/IEC/IEEE 29148:2018.

- An **SRS** is one document that states all the software requirements of a release, agreed by client and team.
- ISO/IEC/IEEE 29148:2018 lists what an SRS should contain; teams tailor the outline and drop empty sections.
- Needed most when a contract, a regulator or a distributed team depends on a written agreement.
- In an agile team the SRS stays **short**: context, rules and non-functional requirements live in the SRS, while the details of each feature live in user stories in the backlog.

⚠ An SRS says *what*, not *how*. Class names, file formats and algorithms belong in the design (Chapter 06) unless they are real constraints.

An SRS excerpt for ITC Club Hub

```
# ITC Club Hub · Software Requirements Specification
Version 0.3 · 2026-10-13 · Team Angkor

## 1 Introduction
### 1.1 Purpose
States the requirements of Club Hub release 1 for the team,
the client (Student Affairs Office) and the testers.
### 1.2 Scope
Approved clubs publish events; students register, cancel
and check in. Out of scope: payments, chat, room booking.
### 1.4 Definitions
Registration: a student's reserved place in one Event.
Check-in: recording that a registered student attended.

## 3 Specific requirements
### 3.2 Functions
- FR-01 The system shall let only an Approved club
  publish an Event with title, start, location, capacity.
- FR-03 The system shall reject a registration when the
  number of Registered students equals the capacity.
- FR-05 The system shall reject a registration for an
  event whose start time has passed.
### 3.4 Performance
- NFR-01 Check-in shall confirm within 1 s for 95 % of
  scans when 100 students check in within 10 min.
### 3.7 Security and privacy
- NFR-06 The phone number shall be optional and visible
  only to organisers of events the student joined.
```

Attribute	FR-03
Priority	Must
Source	N-02 (organiser), safety rule (administrator)
Rationale	The room limit is a safety rule
Verification	Unit test `RejectsWhenEventFull`
Stories	US-02

- **shall** marks a binding requirement; avoid "should" and "may" in requirement text, they hide whether it is mandatory.
- One requirement per id; ids are never reused, even after deletion.
- Versions and dates on top; the file lives in Git next to the code.

A requirement quality checklist

Quality	Question to ask	Club Hub failure
Unambiguous	Can two readers understand it differently?	"Cancel up to one day before": calendar day or 24 hours?
Complete	Are all cases, inputs and errors covered?	Nothing says what happens when the event is full
Consistent	Does it contradict another requirement?	FR-06 says 24 h, the reminder text says "cancel until the start"
Testable (verifiable)	Can a test pass or fail against it?	"The system should be fast"
Singular	Does it state exactly one thing?	"Register, get a reminder and see the map"
Feasible	Can we build it with our time and stack?	"Face recognition at the door" in a C++ CLI in two sprints
Necessary	Does a stakeholder need it? Can we name the source?	A leaderboard of the most active students, asked by nobody
Traceable	Has it an id, a source and a verification method?	A requirement in a chat message, with no id

☑ ISO/IEC/IEEE 29148:2018 names nine characteristics for a single requirement (necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct, conforming) and five for a whole set (complete, consistent, feasible, comprehensible, able to be validated).

Weak words to hunt for

`fast` · `easy` · `user-friendly` · `flexible` · `secure` · `as appropriate` · `etc.` · `and/or` · `should` · `normally` · `many`

Each one hides a number, a condition or a decision that someone has not made yet.

"The system should be fast" → a testable requirement

1. **Which operation?** Ask the organiser: the check-in at the door (N-03).
2. **What is measured?** Time from the scan to the confirmation on screen.
3. **Which value, for how many cases?** 1 second for 95 % of scans (a percentile, because one slow scan is acceptable, a slow queue is not).
4. **Under which load?** 100 students checking in within 10 minutes, 500 registrations stored.
5. **How is it verified?** A test replays 100 scans and computes the 95th percentile.

☑ **NFR-01** Check-in shall confirm within 1 second for 95 % of scans when 100 students check in within 10 minutes. *Verification: load test.*

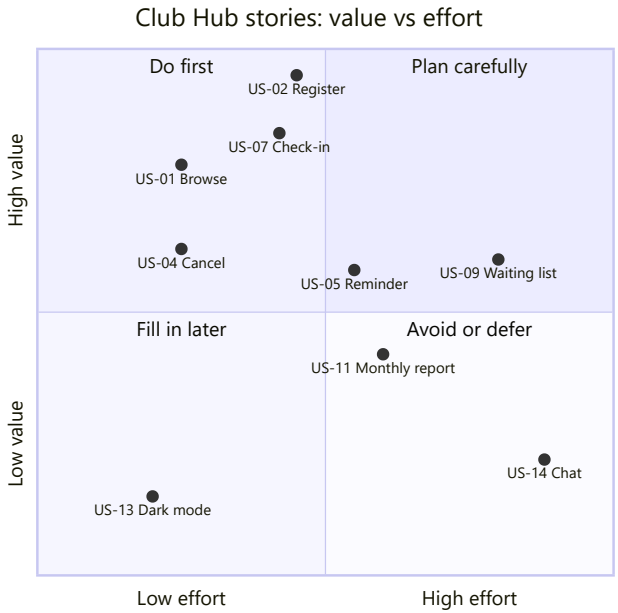
Bad	Testable rewrite
The app shall be user-friendly.	4 of 5 first-time students register for an event in under 60 s without help.
Data shall be secure.	Only organisers of an event can open its attendance list; anyone else gets "Access denied".
It shall handle many events.	It stores 200 events and 5,000 registrations per semester and loads them in under 2 s.
Reminders are sent in time.	A reminder is sent between 24 h and 23 h before the event start.
Students can cancel etc.	FR-06 and FR-07 (slide 5): one rule per requirement, no "etc.".

Prioritisation: MoSCoW and value versus effort

MoSCoW	Meaning for release 1
Must	Without it the release is useless or illegal (register, capacity rule, check-in)
Should	Important, painful to leave out, but a workaround exists (waiting list, reminder)
Could	Nice to have, dropped first when time runs short (CSV export, dark mode)
Won't (this time)	Agreed as out of scope for this release (chat with the organiser)

MoSCoW comes from DSDM (Agile Business Consortium). Its guideline: Must stories take **at most about 60 %** of the effort, so that the Should and Could stories absorb the estimation errors.

Value vs effort ranks inside each MoSCoW class: high value and low effort first.



The Club Hub backlog with MoSCoW and story points

Id	Story (role · capability)	MoSCoW	SP	Requirements
US-01	Student · browse upcoming events	Must	3	FR-02
US-02	Student · register for an event	Must	5	FR-03, FR-04, FR-05
US-03	Club leader · register a club	Must	3	FR-10
US-04	Student · cancel a registration	Must	3	FR-06, FR-07
US-06	Administrator · approve or reject a club	Must	2	FR-11
US-07	Organiser · check a student in at the door	Must	5	FR-08, NFR-01
US-08	Club leader · publish an event	Must	3	FR-01
US-05	Student · receive a reminder 24 h before	Should	5	FR-09
US-09	Student · join the waiting list of a full event	Should	8	FR-12
US-10	Organiser · see attendance of an event	Should	3	FR-13
US-11	Administrator · monthly activity report	Should	5	FR-14
US-12	Organiser · export attendance as CSV	Could	2	FR-15
US-13	Student · dark mode	Could	2	NFR-09
US-14	Student · chat with the organiser	Won't	-	-

Check the Must share

Must = 3 + 5 + 3 + 3 + 2 + 5 + 3 = **24 SP**

Should = 5 + 8 + 3 + 5 = 21 SP · Could = 4 SP

Planned total = 24 + 21 + 4 = 49 SP

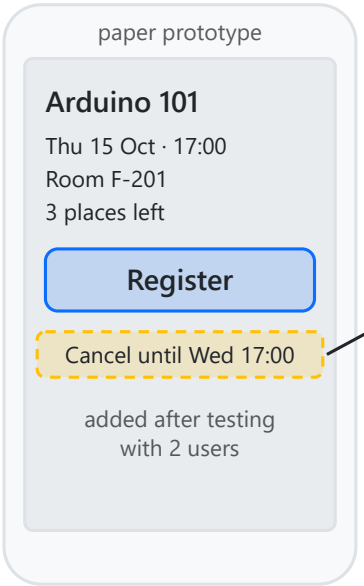
Must share = 24 / 49 ≈ **49 %**, below the 60 % guideline.

- **Story points** are relative sizes (1, 2, 3, 5, 8, 13), estimated together, for example with planning poker (Chapter 07).
- With a velocity of about 15 SP per sprint, the Must stories fit Sprint 1 and part of Sprint 2.
- The client decides MoSCoW; the team decides the points.

Validation: are these the right requirements?

- **Validation**: are we building the right product? **Verification**: are we building the product right? (Boehm)
- **Review**: stakeholders or a partner team read the requirements with a checklist; an *inspection* is the formal version with roles and a defect log.
- **Prototype**: users try a sketch of the flow; missing steps show up immediately.
- **Test cases**: writing the test first exposes vague or incomplete requirements.

💡 Each technique finds a different kind of defect, so use all three before the baseline.



Review (partner team, checklist)

Finds "fast" in a draft NFR and a reminder text that contradicts FR-06: ambiguity and inconsistency.

Prototype (2 students try the flow)

Both ask "until when can I cancel?": a missing requirement, the deadline must be shown before registering.

Test case (written before the code)

RejectsLateCancellation: is exactly 24 h before the start still allowed? The rule must say "at least 24 h".

An acceptance criterion becomes a GoogleTest skeleton

```
Scenario: Event full
  Given event "Arduino 101" has 30 of 30 registered
  When student "e20210002" registers
  Then the registration is rejected
  And the message is "Event full"
```

- **Given** becomes the arrange part: build the objects in the stated state.
- **When** becomes exactly one call on the class under test.
- **Then** becomes the assertions; no assertion, no test.
- The test name comes from the traceability matrix: `RegistrationService_RejectsWhenEventFull` is `TEST(RegistrationService, RejectsWhenEventFull)`.

⧗ Written now as a skeleton, compiled in Sprint 1 when `RegistrationService` exists (Chapter 06 designs it, Chapter 09 tests it in depth).

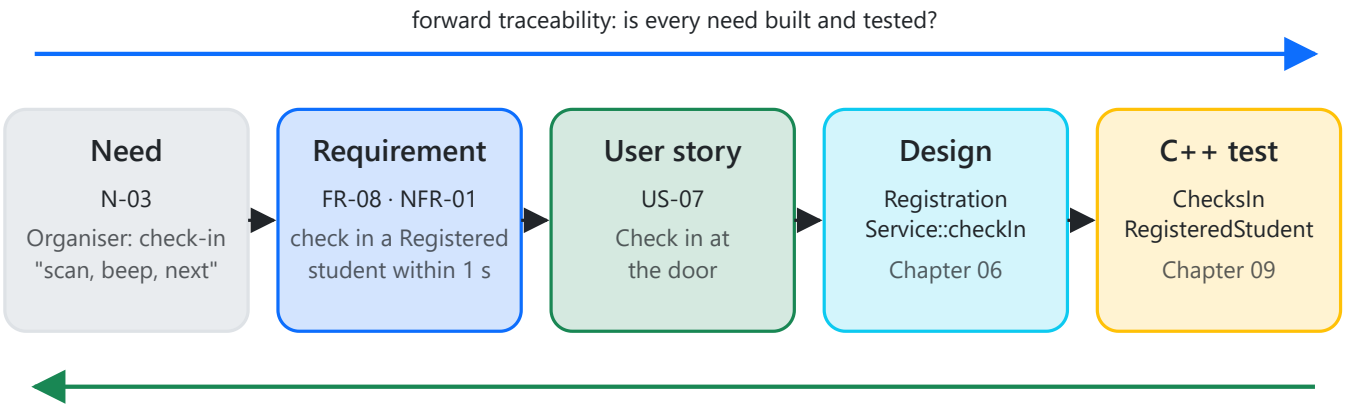
```cpp
// tests/registration_service_test.cpp
#include <gtest/gtest.h>
#include <chrono>
#include <string>
#include "clubhub/registration_service.hpp"
#include "in_memory_repositories.hpp"

using namespace clubhub;
using namespace std::chrono;

// FR-03 · US-02 · scenario "Event full"
TEST(RegistrationService, RejectsWhenEventFull) {
    // Given event "Arduino 101" has 30 of 30 registered
    InMemoryEventRepository events;
    InMemoryRegistrationRepository registrations;
    const auto start = sys_days{2026y / October / 15} + 17h;
    events.save(Event{"E-01", "Arduino 101", start, 30});
    for (int i = 0; i < 30; ++i) {
        registrations.save(
            Registration{"E-01", "s" + std::to_string(i)});
    }
    RegistrationService service{events, registrations};
    const auto now = start - days{3};

    // When student "e20210002" registers
    // Then the registration is rejected with "Event full"
    EXPECT_THROW(service.registerStudent("E-01", "e20210002", now),
                 EventFullError);
    EXPECT_EQ(registrations.countRegistered("E-01"), 30u);
}
```

Traceability: from a stakeholder's need to a C++ test



One row of lab05/traceability.csv links all five ids.

- **Traceability** is the ability to follow a requirement to its source and to everything built from it.
- **Forward**: from need to requirement, story, design and test. Finds needs nobody implemented.
- **Backward**: from code or test to the requirement. Finds *gold plating*, work nobody asked for.
- Kept as a simple matrix (CSV) with stable ids; ids in issue titles and commit messages make the links searchable.

A traceability matrix excerpt

Req	Source	Story	Design element (Ch. 06)	Future test (Ch. 09)	Status
FR-03	N-02, administrator	US-02	`RegistrationService::registerStudent`	`RegistrationService_RejectsWhenEventFull`	Planned
FR-04	Case rule	US-02	`RegistrationService::registerStudent`	`RegistrationService_RejectsDuplicateRegistration`	Planned
FR-05	Case rule	US-02	`RegistrationService::registerStudent`	`RegistrationService_RejectsPastEvent`	Planned
FR-06	President	US-04	`RegistrationService::cancel`	`RegistrationService_RejectsLateCancellation`	Planned
FR-08	N-03	US-07	`RegistrationService::checkIn`	`RegistrationService_ChecksInRegisteredStudent`	Planned
NFR-01	N-03, Lab-02 scenario	US-07	TBD (Lab-06)	`CheckInLoad_P95UnderOneSecond`	Planned
FR-12	President	US-09	TBD	none yet	Gap: no test name

Coverage

Does every Must requirement have at least one story and one test? FR-12 does not yet: fine for a Should, a defect for a Must.

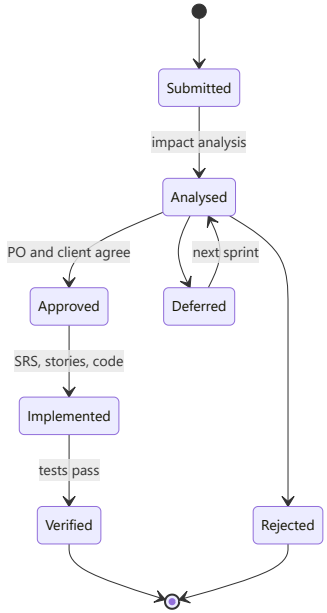
Impact analysis

If FR-06 changes, the matrix lists US-04, `RegistrationService::cancel` and one test to update.

Orphans

A story or test with no requirement row is either a missing requirement or gold plating: ask the Product Owner.

Change is normal: route it through a change request




```
## CR-003 · 48-hour deadline for catered events
Requested by: Robotics Club president · 2026-10-20
Reason: food is ordered two days ahead; late
  cancellations wasted 15 % of the last budget.
Affected (from the matrix): FR-06, US-04,
  `RegistrationService::cancel`,
  `RegistrationService_RejectsLateCancellation`
Impact: +3 SP; Event gets `cancellationHours`
  (default 24); 2 new tests; SRS v1.0 to v1.1
Decision: Approved by PO and client, 2026-10-22,
  planned for Sprint 2
```

- A **baseline** is an agreed, versioned set of requirements (for example SRS v1.0 tagged in Git). Before it, edit freely; after it, change through requests.
- Every request gets an id, a reason, an impact analysis and a recorded decision. Rejected requests are kept.
- Chapter 10 returns to change requests during maintenance.

WRAP-UP

Best practices and common mistakes

✓ Talk to real stakeholders

Interview and observe several roles; a brief written by one person always misses the organiser's or the administrator's view.

✓ Put a number on every quality

Operation, measure, value, load and verification method: "1 s for 95 % of scans" instead of "fast".

✓ One id, one statement, one test

Stable ids, a source for each requirement, acceptance criteria before estimation, and a test name in the matrix.

✗ Solutions disguised as requirements

"Use a MySQL database" or "a blue button" fixes the design too early. Ask "why?" until you reach the real need.

✗ Everything is a Must

When Must stories exceed about 60 % of the effort, one estimation error sinks the release. Negotiate openly with the client.

✗ Write once, never update

An SRS that disagrees with the backlog is worse than none. Baseline it, trace it, and change it only through change requests.

Chapter quiz 10 questions

1. Which of these is a non-functional requirement for ITC Club Hub?

- A student can cancel a registration.
- Check-in confirms within 1 second for 95 % of scans.
- An administrator approves or rejects a club.
- The system rejects a second registration for the same event.

2. What mainly distinguishes a user requirement from a system requirement?

- The user requirement is written in natural language for stakeholders; the system requirement is precise and detailed for developers and testers.
- User requirements are always non-functional.
- System requirements are written after the code is finished.
- User requirements cannot be changed once written.

3. An organiser forgets to mention that walk-ins are admitted when seats remain. Which technique is most likely to reveal this?

- A questionnaire sent to all students
- Reading the SRS template
- Observing check-in at a real club event
- A MoSCoW vote in the team

4. Applied: "As a developer I want a CSV repository class so that data is stored." Which INVEST quality is most clearly missing?

- Independent
- Estimable
- Small
- Valuable

5. In a Given/When/Then acceptance criterion, what must the Then step contain?

- The name of the C++ class that implements the story
- An observable result that can be checked to pass or fail
- The story points of the story
- The role and the benefit of the story

6. Applied: in the textual use case UC-02, a student who is already registered selects Register again. Where is this handled?

- Main success scenario, step 5
- The precondition
- Alternative flow 3a
- The postcondition

7. In an SRS following ISO/IEC/IEEE 29148, where does NFR-01 (check-in within 1 s for 95 % of scans) belong?

- 1 Introduction · Scope
- 2 References
- 3 Specific requirements · Performance
- 5 Appendices · Glossary

8. Applied: which requirement is testable as written?

- The app shall be user-friendly.
- The system shall respond quickly.
- The system shall handle many events.
- 4 of 5 first-time students register for an event in under 60 s without help.

9. Applied: a backlog has 60 story points of Must, Should and Could stories; the Must stories add up to 45 points. What do you conclude?

- Must share 25 %, well balanced
- Must share 45 %, within the guideline
- Must share 75 %, above the 60 % guideline: renegotiate some Musts
- Must share 60 %, exactly at the guideline

10. A change request moves the cancellation deadline from 24 h to 48 h. What does the traceability matrix give you?

- The stories, design elements and tests linked to FR-06 that must be updated
- The number of story points the team completed last sprint
- The list of stakeholders to interview next
- Nothing: matrices are only used for audits

WRAP-UP

Summary

Requirements engineering is a loop: elicit, analyse, specify, validate, manage.	Combine elicitation techniques: interviews for depth, observation and prototypes for tacit knowledge.	A short SRS following ISO/IEC/IEEE 29148 states scope, definitions, FRs and measurable NFRs.
Functional says what, non-functional says how well; user requirements are refined into precise system requirements.	A user story is card, conversation and confirmation; INVEST checks it, Given/When/Then confirms it.	MoSCoW with Must at most about 60 % of the effort; value vs effort ranks inside each class.
Map every stakeholder and resolve conflicts on interests, with the decision recorded.	Textual use cases hold the full dialogue: a main flow plus numbered alternative flows.	Validate with reviews, prototypes and tests; trace need → requirement → story → design → test and change through requests.

Open Lab-05: 5 tasks + 1 challenge

Next chapter: 06 · Unified Modeling Language (UML)