Requirements and specifications

- Requirements generally document what the client/user wants/needs out of a system
- Specifications go further, detailing the technical issues that must also be accounted for – all the constraints on the eventual implementation (firewalls? backups? standards? etc)

Communication/audiences

- Requirements and specifications must each target a wide range of readers and comprehensively define the system, yet still be clear for all readers
- Plain text alone not usually sufficient to communicate the necessary level of detail
- Typically a mix of paragraphs of text, bullet points, charts, images, diagrams, pseudo-code/algorithms, etc
- Need to be iteratively developed and checked for correctness, completeness, consistency, and to ensure they are not ambiguous
- 375 (systems analysis) will focus on requirements, we’ll do specs

Requirements gathering

- For systems that many people use/interact with, can be a complex process to gather all the data on what the system needs to do
- May use surveys, interviews, group workshops/meetings, observation of existing workflows, documents, and data
- Need to constantly review and cross reference data, looking for conflicts, ambiguities, gaps, etc
- Need different user groups to review the requirements as we write them, to ensure what we're writing is correct

Specifications: top down, modular

- Much like the traditional design approaches, we’ll generally take a top-down and modular approach to describing systems
- First describe system as a whole, then how it is broken down into key components, then describe each component (possibly breaking it into smaller components in turn, etc)
- Using object oriented breakdown/description of the system (as interacting objects, possibly arranged in common patterns) is very similar idea, easily mixed in

Specification document

- Will typically cross-reference with requirements document (and possibly others)
- Generally begins with high level, easily readable intro that describes need for system, key functionality/services, top priorities, and key roles of people and other systems that will interact with system
- Likely to include table of contents, index, glossary, appendices, contact info, etc – we want to be able to find key info easily

While writing specs, keep in mind...

- Analysts need to be able to read it, to verify it matches what the requirements doc
- Project managers, developers, testers, maintainers need to be able to read it, as the basis for their work
- Needs to be kept up to date as changes are made
- Needs to specify desired functionality under normal circumstances, but also behaviour in predictable undesirable events
- Should address quality and performance aspects (standards, response time, load capabilities, bandwith, memory use, etc)

Avoiding ambiguity, misunderstandings

- Need to ensure everyone gets the same idea of what is being described (testers, devs, maintainers, etc)
- Need to avoid use of vague/fuzzy terms when writing, where different readers will have different assumptions about what the term means
- e.g. “system must be fast” – way way too vague: e.g. do you mean response time, number of transactions per minute, ???, and what numerical point do we go from “fast” to “not”?
- Ideally specs should be easily testable: is there some yes/no test we agree on that shows whether system meets requirement

Non-functional requirements

- Some examples of non functional requirements and metrics to make the quantifiable
- Speed: transactions/second, screen refresh time, response time
- Size: number of devices or users supported, M/G/T of storage space required
- Ease of use: training time to learn core tasks [need to identify what tasks somewhere], frequency of reference to help pages
- Reliability: mean time to failure, average down time/week

Subroutine specs

- Will often be necessary to write specs for individual methods/functions (e.g. when we write something that others will use)
- Clearly describe purpose (intended use)
- Specify “normal” behaviour, plus exceptions (error handling) and assumptions (things outside the scope of the component)
- Specify preconditions (what needs to be already in place/completed in order for the routine/component to work)
- Specify postconditions (what new is true after the routine/component is finished ... what did it change?)
- Specify inputs (what data does it take in, from where?) and outputs (what data does it generate, sent to where?)

Detail, clarity, quality

- Be sure to clearly specify data contraints/assumptions (assumptions common to an entire group of components or routines may instead be documented at a higher level)
- Be sure to write with all your audiences in mind, actively seek out feedback (and accept it graciously)
- Have documentation standards that are as clear and rigourous as your code standards, and follow them