CMPT 276 Exam Review

Dr. Jack Thomas
Simon Fraser University
Fall 2020

Exam Format

- 3 hours, timed, one attempt.
- Begins on SFU Canvas on December 14th at 7:00pm – as in, *begins at 7:00pm sharp and ends at 10:00pm*.
- Exam requirements mean **it must be completed in parallel by everyone**.
- Exceptions can only be made for medical or extreme time zone reasons, talk to me immediately.
- I will send out a **reminder email** before the exam.

Proctoring?

- I will be **available on Discord** to answer your questions for the whole 3 hours, which is **where I will also post any class-wide announcements** I need to make during the exam.
- There will not be “live proctoring” in the manner of manually checking in over Zoom, since the return on investment seems poor.

Reminder about Security & Academic Integrity

- Canvas logs your IP address.
- Once again, the exam has been written with an open-book format in mind.
- All standard rules about plagiarism and citing apply here as they do in your assignments.
- **Do not collaborate with other students or otherwise accept help during your exam.**
- MOSS will be used to compare submissions, and students have been caught for plagiarising already this term.

Content Overview

- The exam is **cumulative** (meaning it covers content from across the course), but weighted toward after the midterm.
- **There will be coding** questions concerning Java and Android. Consider opening the relevant IDEs to assist you.
- Both the assignments and the project will be drawn from as well.

How to Review Course Topics

1. Consult your notes!
2. Review the lecture slides, including the Midterm Review.
3. Follow up with the recorded lectures where clarification would help.
4. Check out the sample assignment solutions and Dr. Fraser's videos.

9. Requirements Engineering

- **Definition**: The process of establishing the **services that a customer requires** from a system and **the constraints under which it operates and is developed**.
- **User Requirements** (written plainly so both users and developers can understand them, like “Shall generate monthly reports”) vs. **System Requirements** (describing deeper technical requirements that users may not understand or care about, like “access to reports will be restricted to authorized users listed on the management access control list”)

9. Requirements Engineering

- **Functional Requirements** (concrete, specific, and well-defined, like “there must be a search bar on the map screen that filters results”) vs. **Non-Functional Requirements** (higher-level abstract goals, like “runs smoothly”)
- In theory, completeness and consistency of the requirements list is the goal of RE. In practice, imprecision in language and unknown requirements makes this impossible.

10. Requirements Document (Req Doc)

- The **official statement** of the system's requirements.

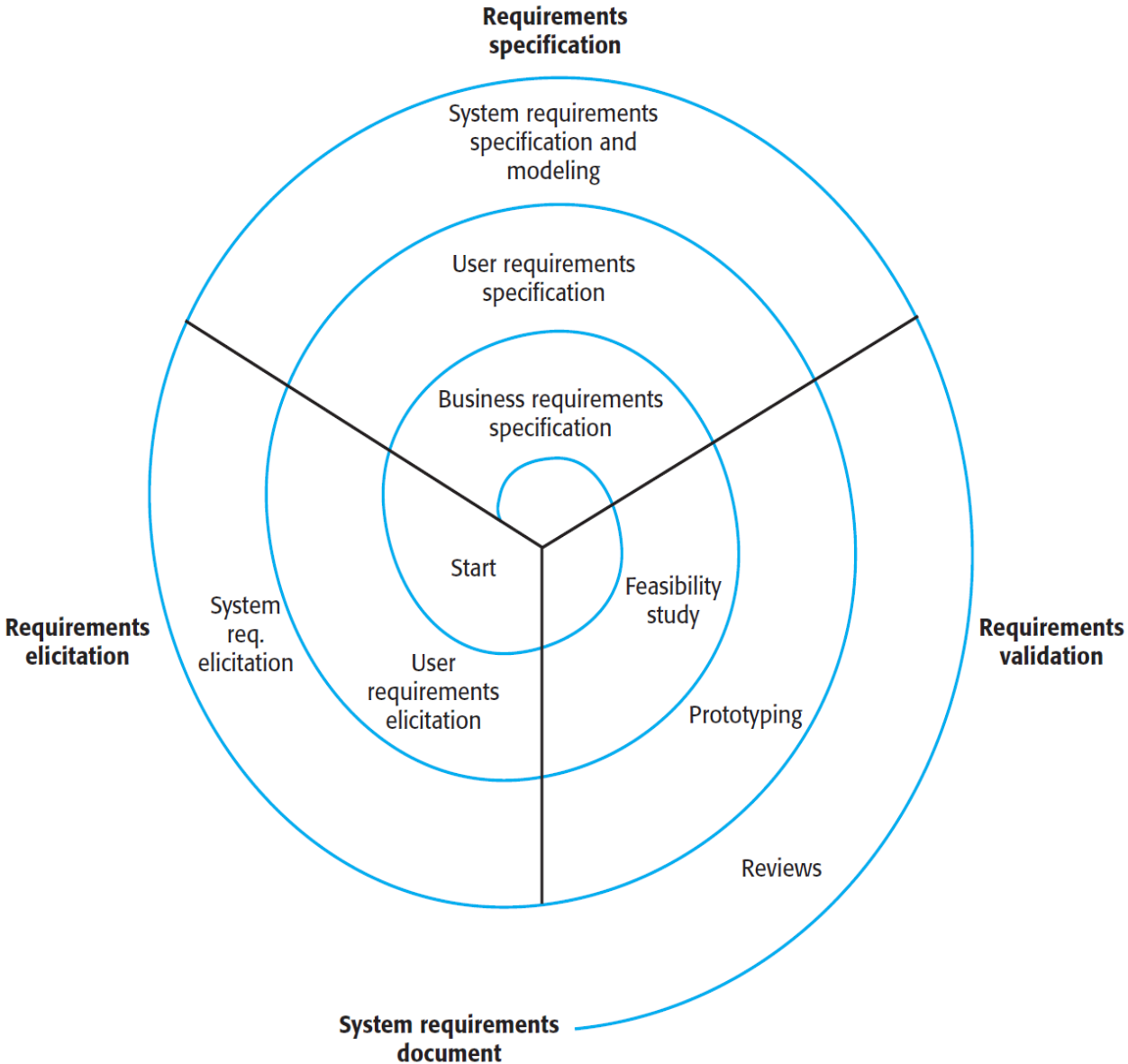
- Typically part of **plan-driven** approaches for large and life-critical systems, **Agile evolves too quickly** for a req doc to be useful.

10. Requirements Document (Req Doc)

- **Specification** is the process of drawing up the requirements document.
- **Design** and specification are in theory separate, in practice **interleaved**.
- Guidelines include using **natural language** in a consistent way that incorporates **domain terminology** and avoids computer jargon.
- Structured or Tabular Specification? Nah.

11. Requirements Elicitation

- Requirements Engineering Process:
  1. Elicitation,
  2. Analysis,
  3. Validation,
  4. Management



11. Requirements Elicitation

- Elicitation involves **requirements discovery**, often through interacting with **stakeholders**.
- **Interviews** involve asking stakeholders for requirements, in either a **closed** (pre-determined question list) or **open** (letting the stakeholder guide the conversation) format.
- **Ethnographies** involve immersing yourself in the stakeholders' **workflow** to better understand their **implicit knowledge**.

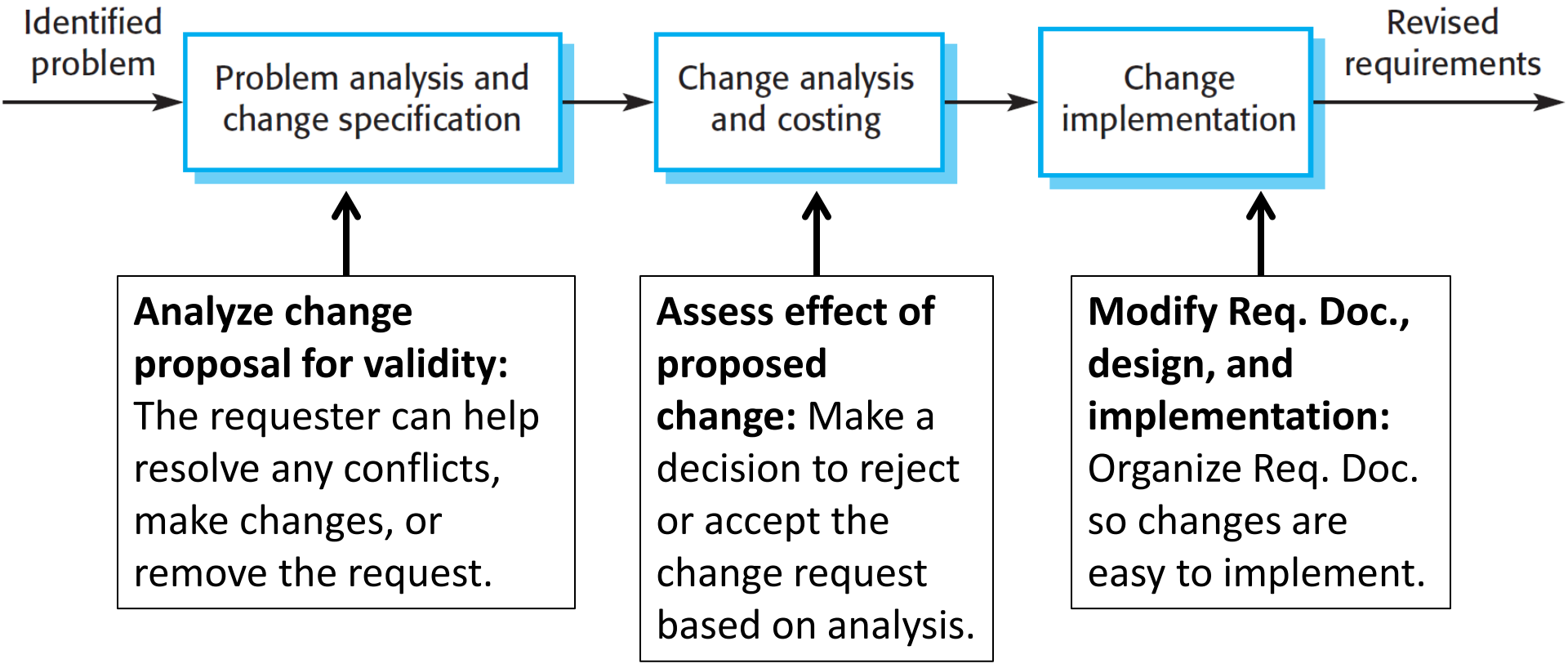
11. Requirements Elicitation

- **User Stories** are a Scrum product that capture product requirements in the format “As [user role], I want [what], so that [why]”.

- **Epic Stories** are too high-level to complete in one iteration, must be broken down into smaller ones.

11. Requirements Elicitation

- **Requirements Management** is how you deal with changing requirements.

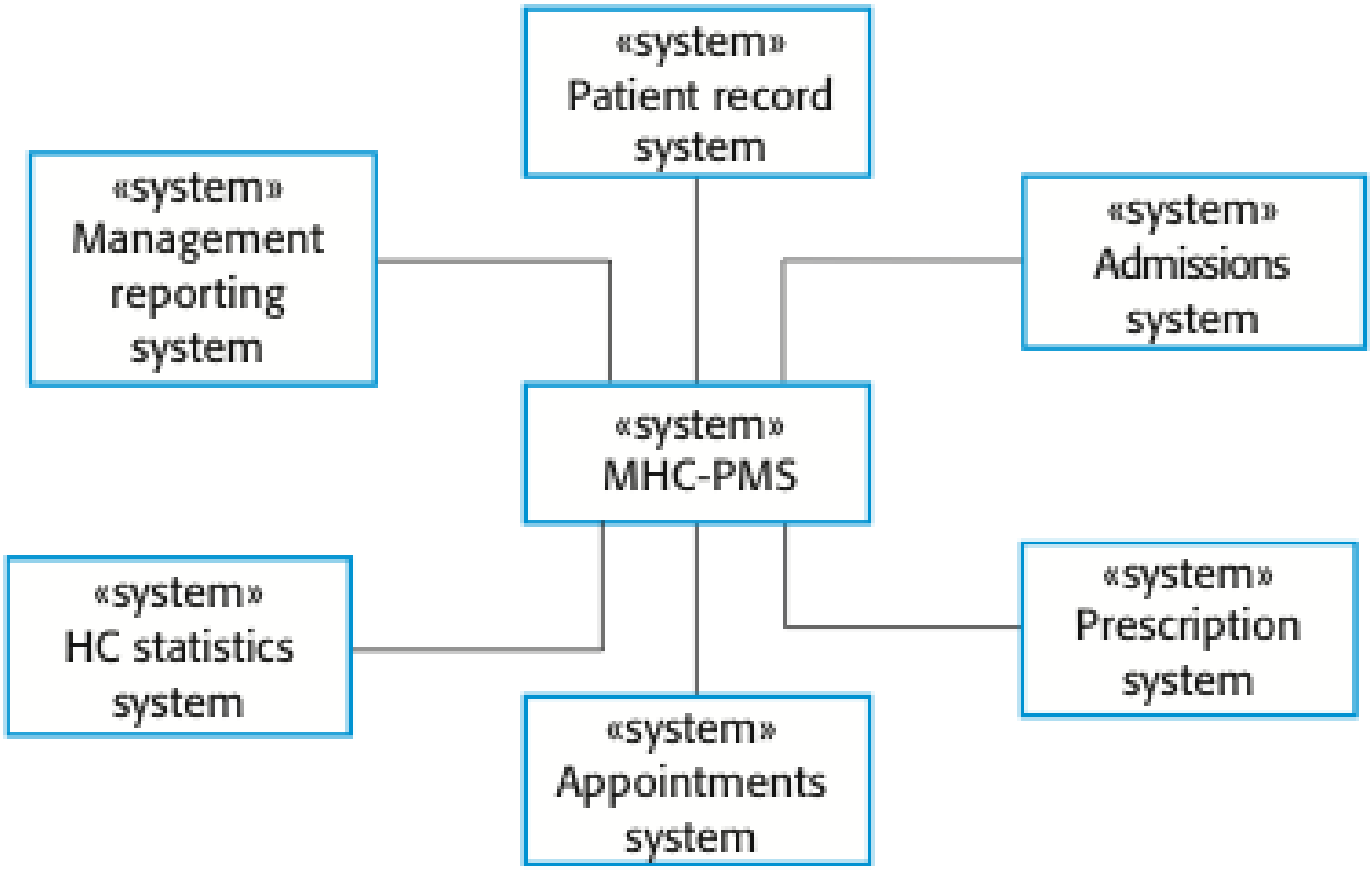


12. System Modelling

- The process of developing **abstract models of a system**, where each model shows **a different perspective of the same system**.
- Perspectives:
  1. External (context)
  2. Interaction (use case)
  3. Structural
  4. Behavioural

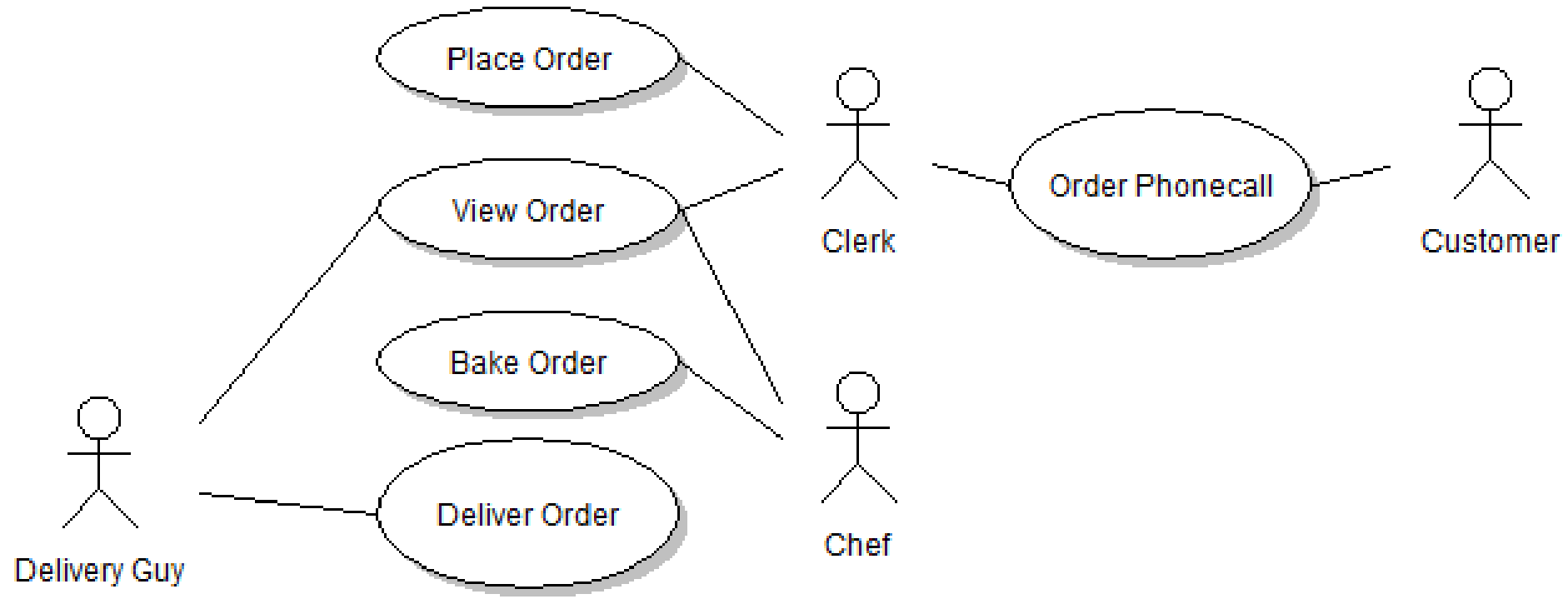
12. System Modelling

- Context Model (External)



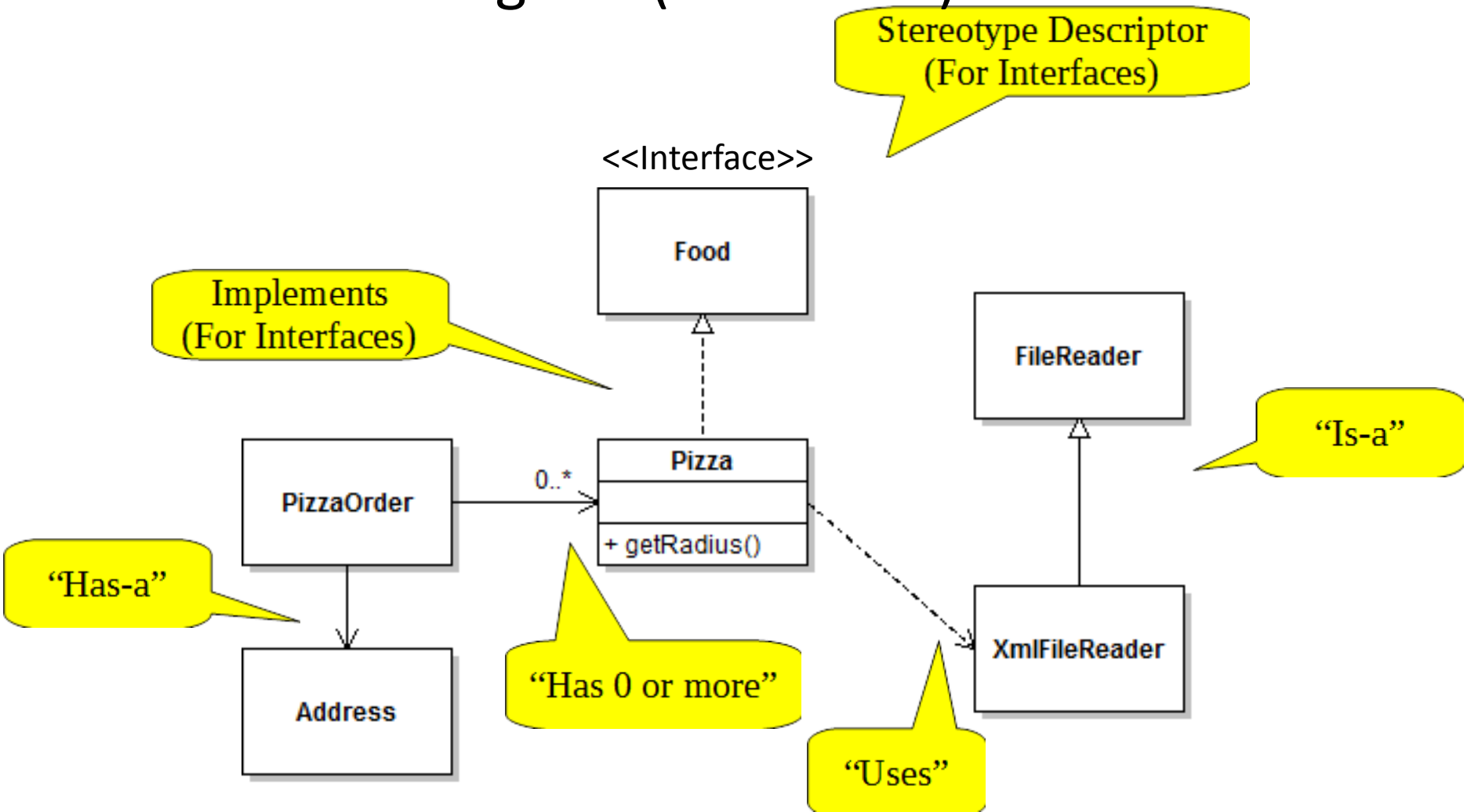
12. System Modelling

- Use-Case Diagram (Interaction)



12. System Modelling

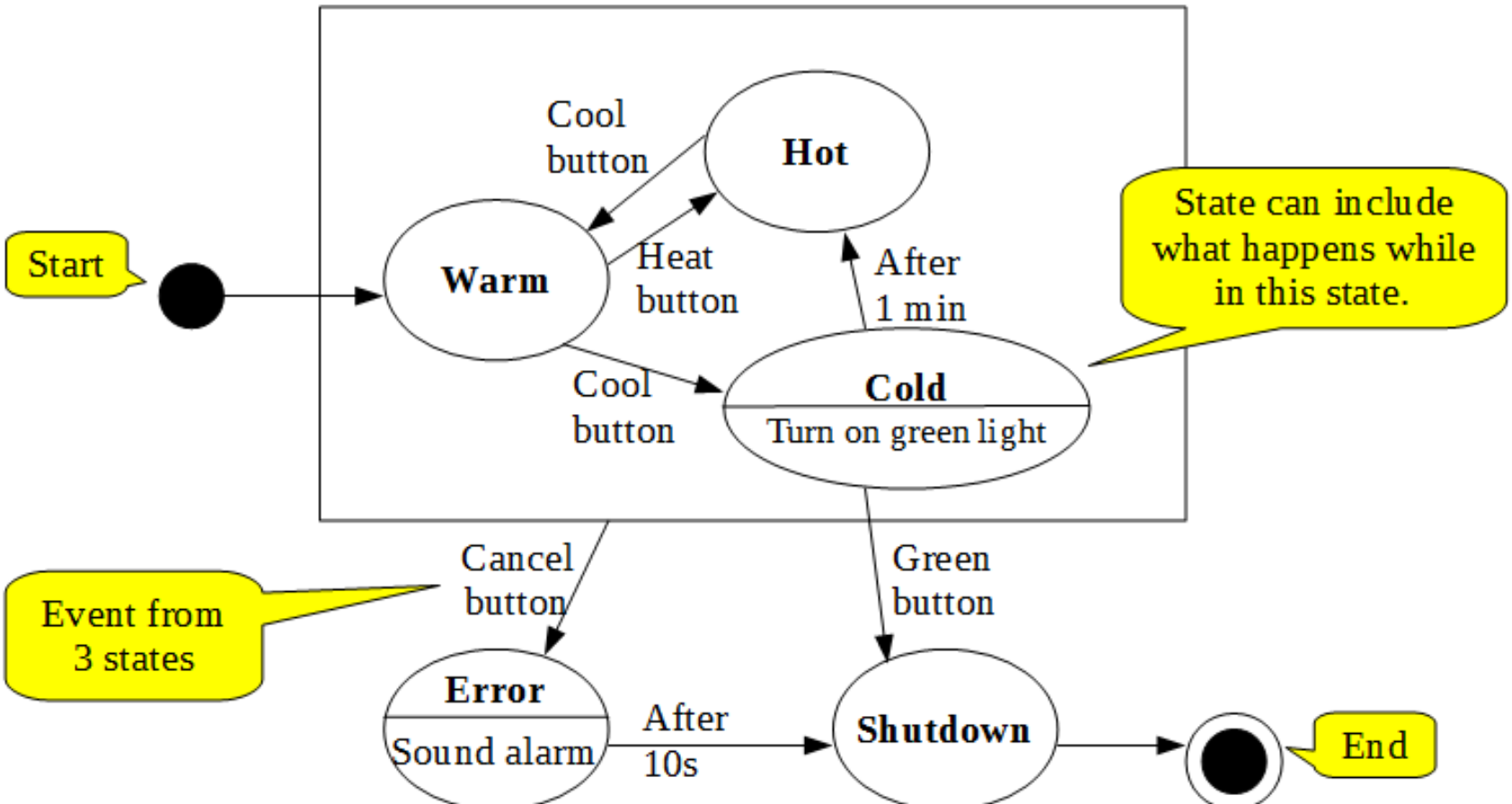
- UML Class Diagram (Structural)



12. System Modelling

- State Diagram (Behavioral)

State diagram for the Acme "Arbitrary Widget"



12. System Modelling

- Understand the **Unified Modelling Language** (UML), particularly for structural class diagrams and behavioural state machines.
- **Model-Driven Engineering**: An approach to software development where models rather than programs are the principal outputs of the development process. Programs are automatically generated from the models.

13. Implementation Issues

- Software's **primary technical imperative** is **managing complexity** through methods like encapsulation.
- **Code Reviews**: Developers checking each others' code for bugs.
  - **Informal**: Any time the code's author walks someone through how the code works.
  - **Formal**: A scheduled, line-by-line review with a checklist of defects to look for.

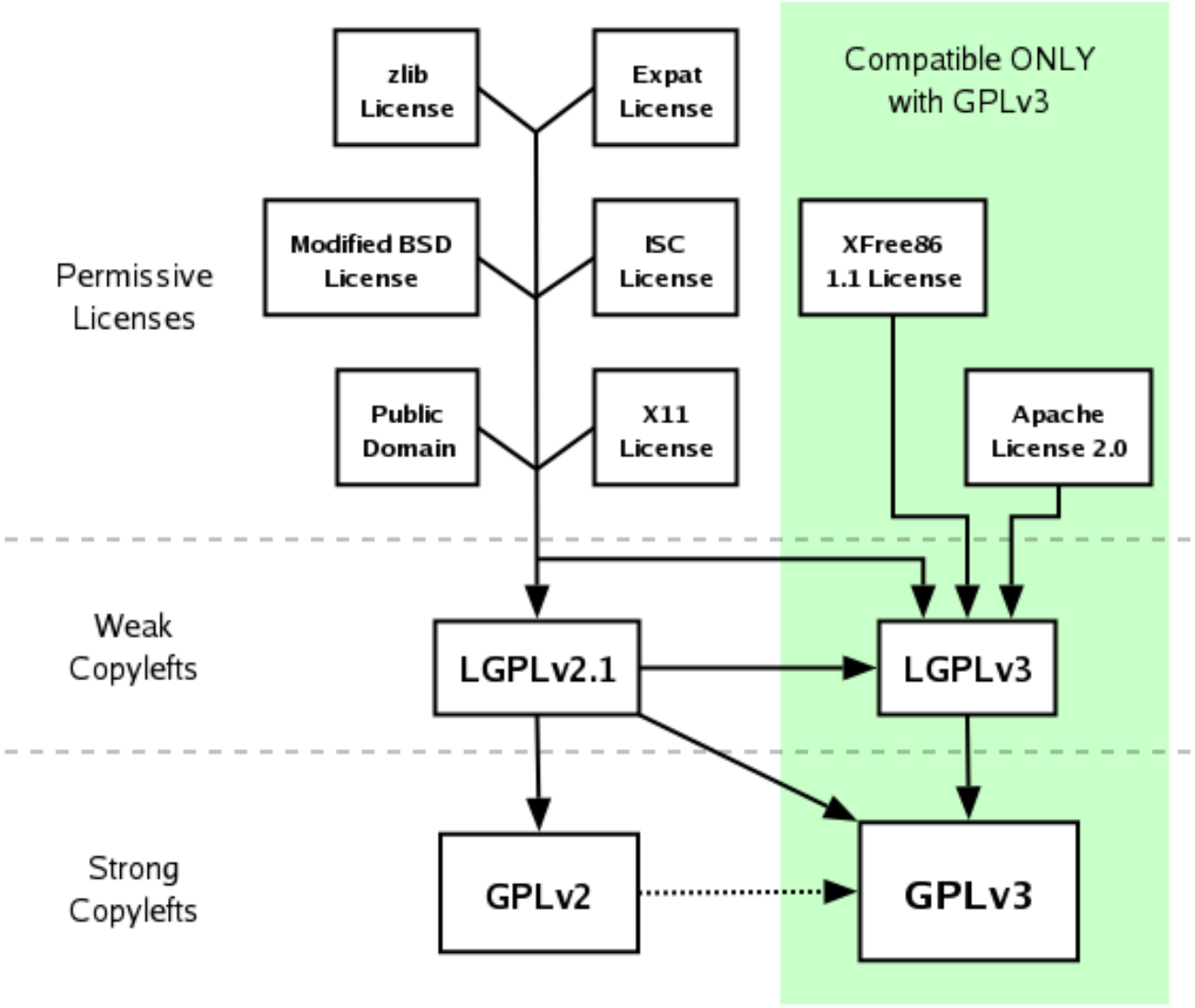
13. Implementation Issues

- **Coding style guidelines** are meant to alleviate complexity by removing one more thing to worry about. Style differences are often matters of **personal preference**, **consistency** is more important than any one decision.
- **Reusing code** is a standard development practice, but **it isn't free** – needs to be integrated into the new system, and there's a danger of trusting old code too much even though **you may uncover new bugs** that weren't possible in the previous system.

14. Legal and Ethical Issues

- **Open-Source** is publicly-available code released under one of several possible **licenses**:
  - **GPL**: If included in a project, that whole project must also be released under GPL.
  - **LGPL**: Can use without making changes without releasing the rest of the project.
  - **BSD/MIT**: Can fully incorporate and change without releasing the rest of the project.

14. Legal and Ethical Issues



14. Legal and Ethical Issues

- **Non-Disclosure Agreements** (NDAs) are common parts of employment contracts in the tech industry used to protect proprietary information.
  - **Non-Compete clauses** can limit your ability to work in a field even after you leave the job.
  - NDAs are **not meant to protect illegal activity**, but they **shift the burden of proof onto the whistleblower**.

14. Legal and Ethical Issues

- **Professional Ethics**: Standards of conduct expected in a professional community above and beyond the letter of the law. Typically concern competency, confidentiality, and honesty.
- Professional organizations in the software field include the **ACM** and **IEEE**, which publish their own **code of ethics**, but have **limited ability to police the industry**.

15. Social and Economic Issues

- Professional ethics prefers to consider a limited set of problems that **ignores systemic issues**.
- Software developers are **workers** who trade their labour for wages and rarely share the profits, startup and corporate culture obscures this.
- There are **many ways your work can be exploited**, but this exploitation is not the product of individually immoral managers, it is **the natural result of unregulated market pressures**.
- To fight for your rights and livelihood, **engage in labour activism** and **build solidarity with your fellow workers**.

15. Social and Economic Issues

- The tech industry has a **significant demographic skew** in terms of race, gender, and class.
- This can result in **narrow perspectives** that can **reproduce bias within technology** itself, and **tensions with local communities** who host a closed and insular tech industry.
- Women have a history of accomplishment in CS despite also regularly being marginalized, and their **representation in the field today is low even for STEM**. The resulting **unwelcome and even unsafe environment** is everyone's responsibility to address and change.

15. Social and Economic Issues

- The tech industry has many controversial impacts on society, including **automation of labour, surveillance, the gig economy, autonomous weapons/policing, and social media**. The field bears a **collective responsibility** for the social consequences of these technologies.
- There is a **history of idealism and radicalism in tech**, including the Open Source community, hacktivism, cryptograph and privacy advocacy, and more.

15. Social and Economic Issues

Image Credit: It's still Sega's Sonic the Hedgehog what do you want from me.

16. Software Design Patterns

- A description of a **common software design problem** and the **essence of its solution**.
- Commonly traced to the **“gang of four” and their book** on design patterns, laying out 23 patterns and 3 categories (creational, structural, behavioral)
- Design patterns are **best practices** for situations that commonly occur in **object oriented programming**.
- They are **abstract**, **do not have a single universal format**, and there is **no one comprehensive list**.

16. Software Design Patterns

- The **Observer** pattern is an example where a subject class maintains a list of observers to be updated whenever an event occurs, which usually includes a way to attach and detach observers and send out an update.
- The **Publisher/Subscriber** pattern is similar, but instead has a broker class that acts as a middleman for the posting and receiving of updates.
- While patterns are usually generalized, different domains make more use of some than others, generally requiring a developer to **specialize in their field's preferred patterns**.

17. Teamwork and Professional Practices

- **Groups** are any collection of coworkers, **teams** are in a social state of cooperation.
- **Advantages**: Tackle larger projects, higher quality, learn more, react faster, sense of commitment.
- **Life Cycle**:
  - **Formed**: First impressions.
  - **Early Days**: First obstacles emerge.
  - **Normalized**: Team settles into a rhythm.
  - **Productive**: Produces the bulk of your work.
  - (Bonus) **Ascendant**: Temporary peak output.
  - **Adjourning**: Winding down a team.

17. Teamwork and Professional Practices

- **Tips**: Mutual respect is fundamental, be constructive, listen to criticism, keep your commitments, communicate, create an agreeable decision-making process that respects disagreements but doesn't avoid conflict.
- Teamwork **isn't always possible** with every group, recognize when a team is failing to form and try to intervene, but also **be prepared to cover your own work individually**.
- As for the job-related material, the exam for that comes **after you graduate**.