MT2 study questions

Software Requirements and Specifications (Concordia University)

SOEN342 - Midterm 2 Study

Lecture 9: Intro to Formal Specification

1. For the various stakeholders involved in a project, what are the main advantages and disadvantages of using formal specifications?

Def: A formal software specification is a statement expressed in a language whose vocabulary, syntax, and semantics are formally defined.

Pros	Cons
Reduces ambiguity	Lack of scope, too focused (They do not capture properties of interest for all stakeholders in the project)
Machine-processable	Not flexible, can't be used with Agile
Better support for automatic requirements validation and verification.	Complex
	Not cost-effective

2. What does it mean when two requirements statements are "logically equivalent"?

Two statements are logically equivalent if they have the same T/F values for all cases, that is, if they have the same truth tables.

p	q	$p \rightarrow q$	$\neg q$	$\neg p$	$\neg q \rightarrow \neg p$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

3. What does it mean when a requirements statement is "satisfiable"?

A proposition is satisfiable if there is at least one true result in its truth table, valid if all values it returns in the truth table are true

Satisfiability

A propositional formula p is satisfiable if there is some interpretation I such that $val_I(p) = T$. The satisfying interpretation I is called a model of p, written $I \models p$

4. What does it mean when a requirements statement is "unsatisfiable"?

Unsatisfiability

A formula p is unsatisfiable (or contradictory) if it is false (F) in *all* interpretations.

5. What is the impact of the fact that predicate logic is semi-decidable on requirements specifications?

Definition

A predicate is a declarative sentence whose T/F value depends on one or more variables. In other words, a predicate is a declarative sentence with variables, and after those variables have been given specific values, the sentence becomes a statement.

Decidability sound and complete and terminating

6. Will a completely formal specification solve the problem of correctly addressing the user's needs?

No as their scope is limited, They do not do a good job of specifying user interfaces and user interaction.

7. Can you combine formal specifications with non-formal approaches, like use cases? Why (and how) would you do this? What would be the advantages and disadvantages?

Yes

Lecture 10: Formal Specification with Z

Def: The Z notation is a formal specification language used for describing and modelling computing systems. It is targeted at the clear specification of computer programs and computer-based systems in general.

$$[\text{NOMBRE}, \text{FECHA}]$$

AgendaCumple

$$\begin{array}{l} contactos : \mathbb{P}\, NOMBRE \\ cumple : NOMBRE \pfun FECHA \\ \hline contactos = \operatorname{dom} cumple \end{array}$$

IniciarAgendaCumple

$$\begin{array}{l} AgendaCumple \\ \hline cumple = \emptyset \\ contactos = \emptyset \end{array}$$

AgregarCumple

$$\begin{array}{l} \Delta AgendaCumple \\ nombre? : NOMBRE \\ fecha? : FECHA \\ \hline nombre? \notin contactos \\ cumple' = cumple \cup \{(nombre? \mapsto fecha?)\} \end{array}$$

BuscarCumple

$$\begin{array}{l} \Xi AgendaCumple \\ nombre? : NOMBRE \\ fecha! : FECHA \\ \hline nombre? \in contactos \\ fecha! = cumple(nombre?) \end{array}$$

Recordatorio

$$\begin{array}{l} \Xi AgendaCumple \\ hoy? : FECHA \\ tarjetas! : \mathbb{P}\, NOMBRE \\ \hline tarjetas! = \{n : NOMBRE \mid cumple(n) = hoy?\} \end{array}$$

Δ prefix modifier: indicates an operation that changes (some) state variables in a schema
Ξ prefix observer: indicates read-only operation (query state) of a schema
? suffix input variable, e.g., *name*?
! suffix output variable, e.g., *date*!
' suffix used in postconditions to indicate state change, e.g., *birthday'*

Syntax

1. Explain the core features of the Z specification.

- No limit on the number of entries
- No implied order of entries
- No restriction on format

Z is a mathematical framework in which to:

- Describe systems (as state machines) abstractly yet precisely
- Compose a system out of small pieces
- Use old specifications to build new specifications
- Reason about properties of a system
- Relate views of a system

2. What is meant by the statement that Z provides "state-based" specifications?

Specify state of a system at defined snapshots

State (at a certain snapshot)

$known = \{ David, Judy, Robert \} \in \mathbb{P} NAME$
$birthday = \{ David \mapsto 24\text{-}June, Judy \mapsto 26\text{-}August, Robert \mapsto 8\text{-}July \}$

3. What are the pros/cons of using Z for software specifications?

Pros	Cons

5. In an operation schema, what happens when a precondition is not satisfied? What can

you say about the system's state in this case?

Pre-condition: a necessary condition on input variables
If the precondition is not met, then the system's state cannot change and no operations can be applied.

6. How are complex systems specified in Z?

Using schema combination, complex systems can be specified in Z.

How does this work?

Let's combine *AddBirthday* and *Success*:

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$
$birthday' =$
$birthday \cup \{name? \mapsto date?\}$

Success

$result! : REPORT$

$result! = ok$

$AddBirthday \land Success$

AddBirthdayAndSuccess

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$
$result! : REPORT$

$name? \notin known$
$birthday' = birthday \cup \{name? \mapsto date?\}$
$result! = ok$

General Approach

To combine multiple schemas:

1. Merge all declarations from individual schemas
2. Join all assertions with the logical connective(s)

Example

For $NewSchema = (Schema1 \land Schema2) \lor Schema3$ we obtain the specification

NewSchema

All declarations from Schema1, Schema2, Schema3

$(AssertionSchema1 \land AssertionSchema2)$
$\lor AssertionSchema3$

where *AssertionSchema* represents the conjunction from all assertions in *Schema*.

7. Why are we interested in the "least restrictive" precondition and the "most complete" postcondition? Illustrate the problems that arise when this is not done using examples based on the BirthdayBook specification.

"Least restrictive" precondition, or weakest precondition is desired because it allows for the state to change or operations to be applied on variables. "Most complete" postcondition, or strongest postcondition is desired because it provides a more concise / exact result.

Ex. x > 0 is a weak postcondition compared to x > 5 && x < 20, however x = 10; would be the most useful and strongest postcondition.

Give me some banana bread bro

8. What is meant by "robust" operations when strengthening a specification? How is it done in Z?

Robust operations are operations with added error handling. Using schema combination of error analysis (success cases or known value schemas), the schema becomes robust.

9. Are Z specifications covering functional or non-functional requirements? Explain.

Both. Z specification helps determine any type of specification.

10. What are the issues when you want to design and implement a system based on a Z specification? Could this be done automatically?

Z is a simple mathematical framework in which it describes systems (as state machines) abstractly yet precisely and can compose a system out of small pieces.

However, the issue with Z specification is that it is too formal. This is seen by the lack of relevant tools applying Z specification automatically into code. It remains that Z specification can still be used for automatic implementation.

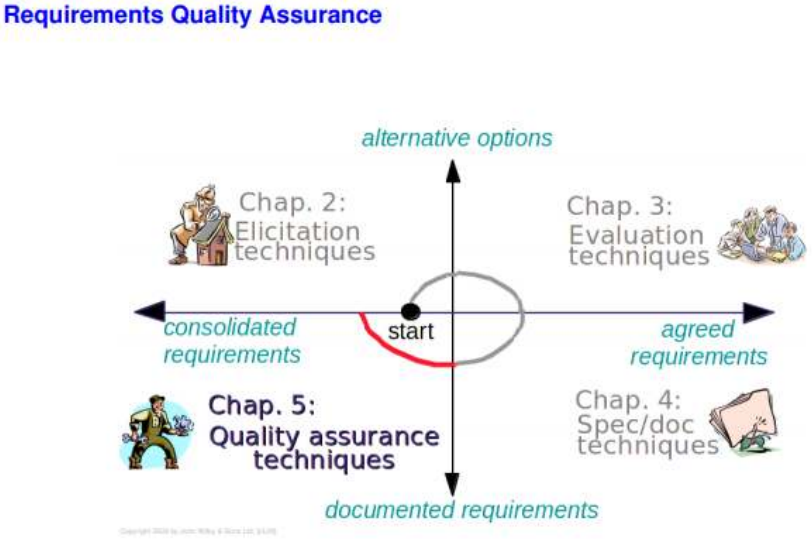
Lecture 11: Requirements Quality Assurance

1. What is the importance of requirements quality assurance (QA)?

- Check if documented requirements satisfy certain qualities
- Might help to reveal defects in the specification
- Derive new requirements through inference

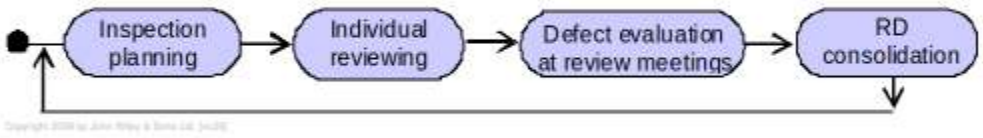
2. At what stage in the software development process do you perform QA?

After documented requirements, in ***consolidated requirements***



3. What artifacts are examined in requirements QA?

4. Explain the QA process, including requirements inspection, review, and consolidation.



Step 1: Inspection

Follow the *WHAT–WHO–WHEN–WHERE* guidelines

WHAT?	Report should contain only facts, not opinions. Constructive, not offensive to RS authors. Use common structure for all inspectors.
WHO?	Inspectors independent from authors. Include different stakeholders (e.g., domain expert, developer, end user). May include QA specialist and RS authors. Minimum size of three.
WHEN?	Not too early (RS still under development); not too late (design & implementation underway). Shorter, more frequent meetings rather than few, long ones.
WHERE?	Critical system aspects (e.g., safety/security), places with already detected defects (and areas impacting/impacted by them).

Step 2: Individual Reviewing

Step 3: Defect Evaluation

Step 4: RD Consolidation

5. Name and explain the different modes for performing inspections.
6. What is the role of defect evaluation at review meetings?

It only to look at defects and report about them:

- ➔ agree on defects (discard false positives)
- ➔ analyse agreed defects
- ➔ recommend actions
- ➔ document conclusions in inspection report

7. How can automatic QA improve a software project? Which types of automatic QA do you know? How are they applied to different specification languages?

Automated QA on NL Specifications

Requires automated processing of natural language text

- using techniques from Natural Language Processing (NLP), Text Mining, and Semantic Computing

We look at a few examples

- NASA's ARM Tool
- Semantic Assistants (Concordia)

Automatic QA can help streamline a software project, is less time consuming than performing it manually and is also less costly.

Needs to be integrated with tools and processes deployed by SOENs

8. Name some of the major guidelines for inspection. What is the motivation behind each of these guidelines?

INSPECTION GUIDELINES

WHAT	Observed facts of inspection and constructed criticism towards RS authors
WHOMST?!	Include all stakeholders and let the inspector be different from the authors
WHEN	Must be after RS and before implementation. Focus on shorter and frequent meetings
WHERE	Focus on the critical parts of the system and places prone to (existing) defects

9. How are checklists used in QA?

DEFECT-BASED CHECKLIST

Omission	Checking for missing information, such as

	relations between requirements or definition of requirements vocabulary
Contradiction	Check for consistency between statements
Inadequacy	Check if it follows stakeholder expectations
Ambiguity	Assure that statements can be interpreted differently, does some words have several meanings?
Immeasurability	Check if criteria are properly measurable and if test data can be derived from it
Noise	Checks for relevance or duplicates
Overspecification	Checks if statement entails premature design choices
Unfeasibility	Check if realistic given constraints such as time, budget, etc..
Unintelligibility	Check if comprehensible by everyone
Poor Structuring	Checks if relevant RD items are covered and if NFRs and FRs are mixed (bad)
Forward Reference	Checks if everything that needs a reference is referenced somewhere
Remorse	Checks if concept has been used before
Poor Modifiability	Checks for dependency between RD items. An RD item should not affect others
Opacity	Dependencies should be explicit

10. Explain the difference between defect-based checklists, quality-specific checklists, domain-specific checklists, and language-specific checklists.

Type of checklist	Definition
Individual Reviewing	The individual checks a given list of questions and issues and review accordingly
Defect-Based Checklists	Check for defects in RD *(Table Above)*
Quality-Specific Checklists	Checks for specific non-functional requirements, such as: - Safety - Security - Performance - usability
Domain-Specific Checklists	Check domain-specific concepts and operation (refinement of domain-independent checklists)
Language-Based Checklists	Based on the specification language used (UML, templates, etc)

11. Name some ways of verifying a Z specification.

1. Checklist for Z (many of these checks can be automated)
2. Syntax Checking: declaration symbols cannot appear in unquantified predicates
3. Type Checking
4. Static Semantics Checking:
    - Each variable must be declared
    - Each variable must have specified initial value
    - Variables must be used within their scope
5. Circularity Checking: No variable can be defined in terms of itself
    Ex. Available = Available \ {bc?}
    This can be automatically detected by finding cycles in the dependency graph

12. With requirements specified formally in propositional logic, how can you show that the specification is consistent?

13. How can you use theorem proving on requirements written in propositional logic to show that a given assertion logically follows from the specification?

14. What are the challenges with automatic theorem proving systems?

15. How can you verify that a specification using decision tables is complete, consistent, and free of redundancies?

16. Name 5 defects common in natural language specifications and explain how to detect them.

17. How can natural language processing (NLP) help in requirements QA?

18. What kind of metrics on requirements specification are computed by NASA's ARM tool?

19. How can the metrics computed by NASA's ARM tool help assess requirements quality?

20. How are the NASA ARM metrics related to the IEEE 830 quality attributes?

21. Name some specific NLP QA analysis services and explain what they do.