Inspection

Document metadata: LO5 5.1 EVI 1

Contents:

Knowledge Base

General Inspection Knowledge

- **Motivation:** For all but the most trivial requirement there will always be a gap between the requirement and what can be achieved through testing. This is a **clear limitation of our test suites.** Review and inspections provide a different approach to quality assurance that is more based on expertise.
- **General idea:** Inspection is a systematic review of software artefacts to find defects and assess quality to gain more confidence that the system behaves as needed.
- **Adequacy of inspection team:** the inspection team consists of one person, the developer of the system. As my developer expertise is similar to the one of a junior developer, it would usually be get at least one software engineer with more expertise onto the inspection team, in a real-world scenario (see Y&P 343).

Inspection and Review Techniques

- **Classic technique:**
  - Follow a checklist while reading the artefact.
  - Checklist contains a set of questions that help identify defects in the inspected artefact and verify that artefact complies with company standards.
  - Modern checklists are structured hierarchically and are used incrementally.
    - Simple checks are used by individual inspectors in the early stages of inspection.
    - Checklists with more complex checks are used in group reviews in later inspection phases.
  - Preface of a checklist should indicate the type of artefact and inspection that can be done with that checklist and the level of expertise required for the inspection.
- **Use-case reading technique** & **Abstraction-driven reading technique:**
  - Adequacy: used to overcome delocalisation in object-oriented programs (can inspect an individual class in an object-oriented program without global knowledge of the program structure).
- **Pair-programming:**
  - Often used in agile software lifecycles where a form of informal inspection occurs during the writing of the software artefact by (in its basic form) letting two programmers work on a single task together.

Choosing an Appropriate Review Technique

We plan to conduct two inspections:

1. Inspection of the testplan created for LO2.
2. Inspection of a software artefact of the system (the scheduling unit).

For (1), we find that a classic inspection technique of using a checklist is appropriate as previous usage for this purpose has been documented as successful (see Y&P 350, for instance).

For (2), we also choose to use a checklist. However, we furthermore plan to deploy an abstraction-driven reading technique. The adequacy of doing so is explained by the component-based nature of the system which strongly resembles an object-oriented nature and allows for overcoming delocalisation.

We follow these steps, adopted to our schedule from the course slides:

1. Plan the inspection by writing a comprehensive checklist.
2. Conduct the inspection and create a review report.
3. Create a list of things to-do.
4. Apply code changes from to-do list.

Testplan Inspection

Checklist Preface

- Artefact inspected: testplan
- Expertise required: junior

Testplan Inspection Checklist

Feature	**How to check**	**Property to check**	**yes**	**no**	**comments**
REQUIREMENTS TO BE TESTED	For each requirement **R**, does the plan include	a priority rating (high/moderate/low)	x		
↑	↑	a justification for the priority rating	x		
↑	↑	a comment on A&T needs (even if "no needs")	x		
A&T NEEDS	*For each item in each A&T needs section, does it include*	a reference to an associated task [T_N] OR direct resolving of the need	x		
TASK SPECIFICATIONS	*For each task, does it include*	a description of what the task involves	x		
↑	↑	stated needs to complete the task	x		
↑	↑	an estimated completion time (ECT)	x		
↑	↑	an indication of which level it is tailored towards (unit/integration/system/research/other)		x	This is not included in the initial version of the test plan.
↑	↑	a justified assignment into a software lifecycle phase	x		
RISK ASSESSMENT & EVALUATION	*Are the following risks addressed*	personnel risks	x		
↑	↑	technology risks	x		
↑	↑	schedule risks	x		
↑	↑	development risks		x	
↑	↑	execution risks		x	
↑	↑	risks from critical requirements		x	
RISK MITIGATION	*Do the following risks have a mitigation plan associated with them*	personnel risks	x		
↑	↑	technology risks	x		

↑	↑	schedule risks	x		
↑	↑	development risks		x	
↑	↑	execution risks		x	
↑	↑	risks from critical requirements		x	
TESTING LEVELS	*Is there at least one task directly associated with testing each of all levels? (unit/integration/system)*	associated system level of task	x		Inferred from the level the associated requirement refers to.
PASS/FAIL CRITERIA	*For each task associated with testing, does it include*	a clearly stated criterion stating the pass condition	x		
↑	↑	a pass condition which implies when the test fails		x	Sometimes, the implied fail condition is not clear.

Testplan Checklist Review Report

Overall, the checklist exposes a few weaknesses of the testplan, including unclear fail conditions for testing tasks, unaddressed risks, and unclear system level of tasks.

As to the unaddressed risks, the testplan does not include them since they were ranked as neglectable for this system and the given tight time schedule. However, this is a clear limitation of the test plan.

Testplan Resulting To-Do

- [ ] Refine the pass conditions so that they clearly imply fail conditions, or consider adding explicit fail conditions.
- [ ] Add an indication of which level a test task is tailored towards.

Scheduling Unit Checklist Preface

- Artefact inspected: scheduling unit
- Expertise required: junior

Scheduling Unit Checklist: comprehensive review, abstract-driven reading

Abstraction for inspection context:

- `result` input always contains the fields used in the artefact, in the needed format
- `addDays()` and `daysBetweenDates()` behave exactly as the method names suggests

Feature	How to check	Property to check	yes	no	comments
DOCUMENTATION	Are the following requirements satisfied?	The unit is documented following JSDoc standards.	x		
↑	↑	The unit documentation contains all required fields (params, returns, what it does)		x	Missing `returns` and description of what unit does.
METHOD DESIGN DECISIONS	Are the following requirements satisfied?	Avoids nested `if` statements		x	See line 27 and check if it can be avoided.
↑	↑	Uses constants where appropriate		x	See line 14 and 15 and check if constant assignments within the try-catch structure would be preferable.
↑	↑	Descriptive method name	x		Descriptive for people working on the system / know its context.

↑	↑	Clear `if` conditions		x	See line 27. Consider adding a comment if condition structure can't be changed
ERROR HANDLING	Are the following requirements satisfied?	Handles invalid input (type/format)		x	Given the abstraction context for this checklist, we can assume that results will be valid input and that the unit does not need to handle invalid input.
↑	↑	Clear about why error is caught if doing so		x	A description of what error is caught / simple console logging may be beneficial.

Scheduling Unit Checklist Review Report

The inspection reveals that the unit performs poorly in certain aspects. It is poorly documented, something that should be prioritised for code which is likely to be read by other people (markers, supervisor) or even maintained by other students (future students building on this prototype). Furthermore, it includes an inexplicable try-and-catch statement, a nested if-statement which might be avoidable, dubious conditions, variables where perhaps constants should be used for code safety. This unit is in need for serious refactoring.

One should not that the checklist does not include any checks on the logic of the unit.

Scheduling Unit Resulting To-Do

- [ ] Add appropriate JSDoc documentation.
- [ ] Change the dubious if-condition to be more descriptive, or add an in-line comment if not possible.
- [ ] Check if you can implement the logic without a nested if-statement. If so, do it.
- [ ] Log any potentially caught errors.
- [ ] Review the possibility of using constants.