06. Software Lifecycle and the LUTF

Software Life cycle discussion

Slide Deck

Download the slide deck from here.

Video Transcript

Hello, this is Amir Shehata with another quick tip on the LUTF.

Today I'd like to propose a new way of working, which could prove advantageous.

Problems with the Traditional Software Development Model

We all learnt the waterfall model at school. I don't know about you, but in practice it is never that clean.

It's a lot more iterative.

As we write the HLD, we find we need to role back and adjust the requirements. The Detailed design is often passed over in favour of going directly to implementation.

The Test Plan is written at the very end.

Bug fixes and testing often result in design changes and even requirement changes.

We endup with a bit of a messy development process.

Under deadlines what inevitably happens is the documentation goes stale.

When the product is then maintained by another developer the documentation (if it exists at all) is sorely out of date making the developer's life hard. Sounds like a familiar story? It does to me.

A solution that definitely doesn't work is introducing more red-tape and documentation. It'll just make the developer's life even harder.

I contend that there exists no zero effort solution. Any solution will need adjusting the development workflow, which will appear like extra effort.

The trick is to reduce the cost of the solution and maximize the benefit.

Proposed Solution

My proposed solution is to bring the test plan development to the centre of the Software Development Cycle.

A well formed requirement must be testable. Therefore we can represent each requirement as a test case.

Each requirement must be designed. The details of the design can also be added to the test case.

We can add extra test cases during the design phase as needed. These test cases could represent new requirements.

When we move to implementation (either after the design phase is completed or in parallel) we can still add extra test cases. Again these test cases can represent new requirements.

This can happen all the way down to the bug fixing phase.

Once the feature is complete we should have a set of test scripts which represent the requirements, test plan and parts of the design.

We can conceivably then extract all the information needed for these documents.

The LUTF Involvement

Let's now imagine that all these test scripts, which include:

- the requirements
- the details on how the Requirements will be designed and
- the test case description

are LUTF test scripts.

We can use the LUTF to automatically extract all this information from the test scripts and generate our documentation.

As you can see we no longer need to keep 3 different documents: A requirements document, an HLD document and a Test plan document.

With this work flow the requirements, design and the test plan are collapsed into one form: the LUTF test scripts. From the test scripts we can generate our documents.

I believe this reduces the overhead and increases our chances of keeping documentation consistent with the implementation.

Caveat

Of course life is always more complicated that it appears on paper.

Diagrams and other mediums might be needed to explain the requirements and the design, which can not be included in a text only form.

However, my argument is that the above process can alleviate much of the required document maintenance.

Let's take the Requirements document as an example.

The first section can have the overview and diagrammatic explanations required. The second section is the table detailing all the requirements.

Updates to the second section of the document can be automatically generated from the test scripts.

This method provides the glue between the code and the documentation. As bugs are fixed or the feature updated, as long as the developer creates new test cases to cover the changes made and regenerate the documentation, the code and the documentation will remain in sync.