Anaplan

Behaviour driven specification

Simon Fraser and Alessandro Pezzoni

19th Overture Workshop, 22nd October, 2021

A sparse problem

According to the International Air Transport Association, there are approximately:

- 9100 airports
- 1100 airlines

Tracking flights every day for a year would require 3.3×10^{13} cells.

However, there are less than 40 million flights made in a year.

It was decided that we needed an alternative engine optimized for sparse problems.

Formal methods were the ideal choice to capture the behaviour of our calculation engines and document any differences, ensuring consistency.

The agile process

- The product owner collaborates with customers to create user stories
- These stories are prioritised
- Development work starts
- Customer feedback is used to create more stories that improve and extend features

When stories are created, the product owner defines a set of *acceptance criteria* (ACs) that determine completion.

Test and behaviour driven development

TDD

- Write failing tests for each change, followed by enough code to make them pass
- Operates at a much finer granularity than user stories

BDD

- Scenarios written at the user level
- Exemplify stakeholder requirements using statements resembling Hoare triples
- Uses a DSL to improve understanding

Our goals

- Specification happens in parallel to the implementation
- Comprehensively specify small chunks of the system at a time
- Express system requirements in a universally accessible form

Adapting the agile process to our needs

Based on the existing engine, we already had a reference for the expected behaviour of the alternative engine.

ACs were going to be the *output* of our process, so a significant portion of our time was devoted to analyzing the behaviour of the existing system.

- Specification and development teams worked on copies of the same backlog, feeding into each other's queues,
- *Executable acceptance criteria* (EACs) were created to document requirements and act as single source of truth
- Completion when all relevant EACs are satisfied

The BDS process

For each behaviour:

- Work with stakeholders to produce a natural language description of the AC
- Write an example scenario, which might require extending the DSL
    - at this stage, the resulting EAC should fail
- Change the specification until this EAC passes

The resulting EACs can then be used by the developers as part of the BDD process.

The outputs of BDS are both EACs and a validated specification.

Example EACs for tic-tac-toe

```
@Eac("When a player places a token, the board is updated")
fun turns() {
    given {
        thereIsABoard(boardA, """
            . | . | .
            . | . | .
            . | . | .
        """)
    }
    whenever {
        playerHasTurn(boardA, O, x = 0, y = 0)
        playerHasTurn(boardA, X, x = 2, y = 2)
    }
    then {
        boardHasState(boardA, """
            O | . | .
            . | . | .
            . | . | X
        """)
    }
}
```

```
@Eac("A player cannot place a token over an existing token")
fun immutableState() {
    given {
        thereIsABoard(boardA, """
            . | . | .
            . | X | .
            . | . | .
        """)
    }
    then {
        playerCannotHaveTurn(boardA, O, x = 1, y = 1)
    }
}
```

The specification adapter

To reaffirm the arguments of (Larsen - Fitzgerald - Wolff, 2011), we chose VDM for its rich tooling and flexibility:

- We can animate the specification either from Overture or programmatically
- We can animate single functions or operations, allowing many entry points into the specification

Instead of using a specialised UI for animation, our adapter transpiles EACs into a VDM module consisting of a single operation.

Results of the process

Documenting system behaviour

- A universally accessible rigorous documentation
- A formal record of differences, all based on informed decisions
- Differences can be documented *before* customers encounter them
- Continuous validation

Lower barrier of entry to contribution

- Easy to analyze a behaviour and write the corresponding EACs
- New members only needed to understand the area affected by their change
- Experienced members could quickly grasp areas they had not worked on previously

The road to adoption

Initially developers were skeptical about the introduction of formal methods:

- Traditionally linked to a waterfall model
- Fear of loss of influence over system requirements

BDS was instrumental in assuaging those doubts:

- Flexible, parallel process
- Iterative process, where specifiers felt like collaborators, not dictators
- The use of a DSL simplified understanding of the requirements

Some metrics

- *200*+ behavioural differences identified, discussed and specified
- *200*+ implementation bugs identified

- *60,000*+ lines of VDM-SL
    - *26,000* over 231 modules for the specification itself
    - 3,000+ VDM unit tests spanning 34,000 lines and 115 modules
- 54 VDM-SL projects
- 7 Java and 1 Rust libraries to aid animation
- 3,000+ EACs

Anaplan