Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Verificação, Validação e Teste de Software (CK0241)

TB05 - Criação de Testes Unitários

Video Splitter

Fernanda Costa de Sousa - 485404
José Douglas Gondim Soares - 485347

Histórico de versões

Versão	Data	Autor	Descrição
1.0	24/11/2022	Fernanda Costa e José Douglas	Criação da versão inicial do documento
1.1	27/11/2022	Fernanda Costa	Atualização do documento: inclusão das seção 1. Introdução 1.1. Aplicação, código fonte e scripts de teste
2.0	28/11/2022	Fernanda Costa	Atualização do documento: inclusão da seção 1.2. Descrição da ferramenta de teste unitário
3.0	29/11/2022	José Douglas	Atualização do documento: inclusão na seção 2 Resultados gerais e 5. Referências
5.0	04/12/2022	José Douglas	Atualização do documento: inclusão na seção 3. Discussão e 4. Conclusão

Sumário

1. Introdução

Inúmeros fatores podem afetar a qualidade do software durante o desenvolvimento, em alguns casos falhas podem causar prejuízos ao projeto como um todo, pois podem gerar problemas e vulnerabilidades nos sistemas e aplicativos.

Os testes unitários podem fornecer um rápido feedback sobre o projeto e a implementação do código. De maneira que ao adotar alguns métodos podemos fazer uma avaliação, identificar possíveis problemas, corrigi-los e trazer mais qualidade ao código.

Este documento tem como objetivo apresentar a criação de testes unitários realizados em alguns requisitos do projeto do aplicativo Video Splitter utilizando testes automatizados.

1.1. Aplicação, código fonte e scripts de teste

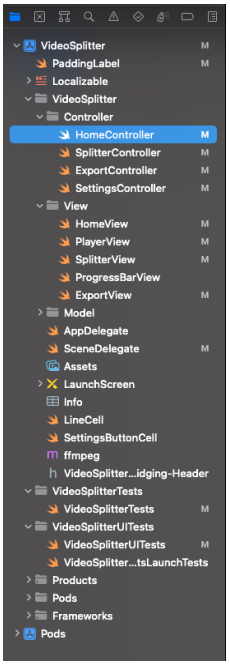
A aplicação Video Splitter é um software de manipulação de vídeos que permite ao usuário cortar um vídeo qualquer em n partes de tamanhos iguais, para então utilizá-las de diversas formas criativas.

Todos os códigos fonte do projeto estão disponíveis em: https://drive.google.com/file/d/1qCbmZgN4tNj_o5mA79KstbxoERbTI3Zh/view?usp=share_link . Os scripts de teste podem ser acessados no seguinte caminho de arquivo: */VideoSplitterTests/VideoSplitterTests.swift*

1.2. Descrição da ferramenta de Teste Unitário

A ferramenta utilizada foi o framework *XCTest*[1], que é uma ferramenta que vem como padrão no *Xcode*[2]. Ao iniciar um projeto podemos selecionar a opção *"Include Unit Tests"* para incluir a possibilidade de fazer testes unitários.

Na hierarquia de pastas do projeto é possível observar que o *Xcode* cria além da pasta padrão do projeto, uma outra pasta na qual estão incluídos os testes da aplicação.



Ao abrir o arquivo padrão de teste disponibilizado pelo *Xcode* podemos ver o seguinte código:

Analisando esse código podemos observar alguns pontos importantes:

- Na linha 7 importamos a ferramenta utilizada em nossos testes, que é a *XCTest* que vem como default no *Xcode*.
- Logo após, temos um *import* com a palavra reservada *@testable*.

O que essa linha faz é importar o projeto para o *namespace* do teste, a palavra chave *@testable* é necessária por causa da forma como o swift opera com seus modificadores de acesso: *Public, Internal e Private*, que permitem visibilidade pública a qualquer namespace, visibilidade somente dentro do mesmo *namespace* e visibilidade apenas da classe/Struct.

Por fim, podemos observar as funções:

- testGenericNumberOfBarsCreated

```
func testGenericNumberOfBarsCreated() throws {
    let splitterController = SplitterController(videoURL: URL(fileURLWithPath:
        "/Users/douglasgondim/Downloads/gabbie.mp4"))
    splitterController.loadViewIfNeeded()

    splitterController.splitValue = 5

    let expectedNumOfBars = Int(ceil(splitterController.totalVideoSeconds / splitterController.splitValue))
    print("expected num of bars: ", expectedNumOfBars)

    splitterController.numberOfVideos = expectedNumOfBars

    splitterController.createNewProgressBars(numberOfVideos: splitterController.numberOfVideos,
        splitValue: splitterController.splitValue)

    print("Num of bars created: ", splitterController.progressBars.count)
    let numBarsCreated = splitterController.progressBars.count
    XCTAssertEqual(expectedNumOfBars, numBarsCreated)


}
```

- testGenericPortionOfVideoPlayed

```
func testGenericPortionOfVideoPlayed(){
    let splitterController = SplitterController(videoURL: URL(fileURLWithPath:
        "/Users/douglasgondim/Downloads/gabbie.mp4"))
    splitterController.loadViewIfNeeded()

    splitterController.currentTime = 16.5
    splitterController.splitValue = 15

    let expectedNumOfVideos = Int(ceil(splitterController.totalVideoSeconds /
        splitterController.splitValue))

    splitterController.numberOfVideos = expectedNumOfVideos

    splitterController.createNewProgressBars(numberOfVideos: splitterController.numberOfVideos,
        splitValue: splitterController.splitValue)

    splitterController.updateProgressBar()


    let currentBarFillValue : Double = splitterController.currentTime / splitterController.splitValue
    let currentFilledBar = Int(currentBarFillValue)

    let currentProgress = Float(currentBarFillValue - currentBarFillValue.rounded(.down))

    var errorFound = false

    //splitterController.progressBars[0].progress = 0.5

    for (idx, bar) in splitterController.progressBars.enumerated(){
        //print("idx: ", idx)
        if(idx < currentFilledBar && bar.progress != 1.0 || idx > currentFilledBar && bar.progress != 0.0
            || idx == currentFilledBar && bar.progress != currentProgress){
            //print("Error found: ", idx, bar.progress)
            errorFound = true
        }
    }
    XCTAssertFalse(errorFound)

}
```

- testNextBar

```
func testNextBar(){
    let splitterController = SplitterController(videoURL: URL(fileURLWithPath: "/Users/douglasgondim/Downloads/gabbie.mp4"))
    splitterController.loadViewIfNeeded()

    splitterController.currentTime = 0.0
    splitterController.splitValue = 10

    let expectedNumOfVideos = Int(ceil(splitterController.totalVideoSeconds / splitterController.splitValue))

    splitterController.numberOfVideos = expectedNumOfVideos

    splitterController.createNewProgressBars(numberOfVideos: splitterController.numberOfVideos, splitValue:
        splitterController.splitValue)

    for _ in 0...expectedNumOfVideos{

    splitterController.updateProgressBar()


    let currentBarFillValue : Double = splitterController.currentTime / splitterController.splitValue
    let currentFilledBar = Int(currentBarFillValue)

    let currentProgress = Float(currentBarFillValue - currentBarFillValue.rounded(.down))


    //splitterController.progressBars[0].progress = 0.5


        for (idx, bar) in splitterController.progressBars.enumerated(){
            print("Bar \(idx): \(bar.progress)")
            if(idx < currentFilledBar && bar.progress != 1.0 || idx >= currentFilledBar && bar.progress != 0.0){
                print("Error found. idx = \(idx), progress = \(bar.progress), current filled = \(currentFilledBar) ")
                XCTAssertFalse(true)
            }
        }
        print("--------------------")
        splitterController.nextVideoSection()
        splitterController.currentTime = splitterController.splitterView!.videoView!.player!.currentTime().seconds
        print("Current time: \(splitterController.currentTime)")
       // splitterController.updateProgressBar()

    }
    XCTAssertFalse(false)


}
```

- testPreviousBar

```
func testPreviousBar(){
    let splitterController = SplitterController(videoURL: URL(fileURLWithPath: "/Users/douglasgondim/Downloads/gabbie.mp4"))
    splitterController.loadViewIfNeeded()

    splitterController.currentTime = splitterController.totalVideoSeconds
    let seekTime = CMTime(seconds:splitterController.totalVideoSeconds, preferredTimescale: 1)
    splitterController.splitterView?.videoView?.player?.seek(to: seekTime, toleranceBefore: CMTime.zero, toleranceAfter: CMTime.zero)

    splitterController.splitValue = 10

    let expectedNumOfVideos = Int(ceil(splitterController.totalVideoSeconds / splitterController.splitValue))

    splitterController.numberOfVideos = expectedNumOfVideos

    splitterController.createNewProgressBars(numberOfVideos: splitterController.numberOfVideos, splitValue: splitterController.splitValue)

    for _ in 0...expectedNumOfVideos-1{

    splitterController.updateProgressBar()


    let currentBarFillValue : Double = splitterController.currentTime / splitterController.splitValue
    let currentFilledBar = Int(currentBarFillValue)

    let currentProgress = Float(currentBarFillValue - currentBarFillValue.rounded(.down))


    //splitterController.progressBars[0].progress = 0.5


        for (idx, bar) in splitterController.progressBars.enumerated(){
            print("Bar \(idx): \(bar.progress)")
            if(idx < currentFilledBar && bar.progress != 1.0 || idx >= currentFilledBar && bar.progress != 0.0){
                print("Error found. idx = \(idx), progress = \(bar.progress), current filled = \(currentFilledBar) ")
                XCTAssertFalse(true)
            }
        }
        print("--------------------")
        splitterController.previousVideoSection()
        splitterController.currentTime = splitterController.splitterView!.videoView!.player!.currentTime().seconds
        print("Current time: \(splitterController.currentTime)")
       // splitterController.updateProgressBar()

    }
    XCTAssertFalse(false)


}
```

2. Resultados gerais

A ferramenta foi utilizada no requisito RF003 e RF004 do documento de plano de V&V. É esperado que em uma versão estável, ela passe em todos os testes, uma vez que só deveriam ser encontrados problemas em caso de mudanças no código que causem efeitos indesejados. Os testes foram capazes de detectar problemas quando erros foram forçados.

3. Discussão

Como apresentado em tópicos anteriores, a ferramenta *XCTest* foi utilizada nos requisitos RF003 e RF004 descritos no documento de plano de V&V.

A ferramenta *XCTest* possui integração com *Xcode* o que facilitou bastante seu uso. Após a análise, concluímos que a aplicação está funcionando da forma esperada.

4. Conclusão

Como apresentado nos tópicos anteriores, o documento aborda a criação de teste unitários da aplicação Video Splitter, realizada a partir da ferramenta *XCTest*. Para dar início aos testes, efetuamos as configurações e preparação da estrutura dos testes. Após uma pequena configuração logo podemos iniciar a implementação dos testes, contudo, a ferramenta demonstrou dificuldade em ligar um conjunto de itens de dados a outros, no mapeamento dos elementos, foi necessário acrescentar valores em forma de mock para prosseguir.

5. Referências

1. XCTest: Ferramenta de Teste Unitário. Disponível em: <https://developer.apple.com/documentation/xctest/>. Acesso em: 27 de novembro de 2022.

2. Xcode: Ferramenta de desenvolvimento. Disponível em: <https://developer.apple.com/xcode/>. Acesso em: 27 de novembro de 2022.

3. Teste unitário com swift – Parte 1. Disponível em: <https://www.redspark.io/teste-unitario-com-swift-parte-1/>.

Acesso em: 27 de novembro de 2022.