AutoRSpec

Dan Shreeve

2017-04-25

Signed declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Daniel Demaine Shreeve

Signature: ....

Date: 03/05/2017

Abstract

Software Testing benefits the development and maintenance of a system by increasing quality and reliability. Software Testing also contributes around half of the cost of producing a system. Automating part or whole of this process reduces the cost while maintaining the benefits. The aim of this project is to produce a system that automatically generates RSpec test cases for model validation in Ruby on Rails. The tests will be generated from a formal database specification that the user has defined using the system. A file can be generated and inserted into the users application and run as though they been written by the user. The time taken to insert the information should be less than the time taken to write the tests manually, otherwise the user does not benefit.

Acknowledgements

First and foremost I would like to thank my supervisor **Dr. Gordon Fraser** for accepting my proposed project and providing impeccable advice and guidance throughout the process.

Finally I would like to thank my parents, **Linda Shreeve** and **Paul Shreeve**, and my grandparents **Elsie Marsden** and **Ray Marsden** for giving me the oppurtunity to go to university and supporting me throughout.

Contents

Title page **i**

Signed declaration **ii**

Abstract **iii**

Acknowledgements **iv**

1 Chapter 1: Introduction **1**
1.1 Importance of Testing 1
1.2 Benefits of Testing and Automation 1
1.3 Ruby on Rails 1
1.4 Project Aims 2
1.5 Project Limitations 2

2 Chapter 2: Literature Review and Research **3**
2.1 Tools to use 3
2.2 Testing and Automation 6
2.3 Testing in Rails 8
2.4 wrapup 9
2.5 Evaluation 9

3 Chapter 3: Requirements and Analysis **9**

4 Chapter 4: Design **9**

5 Chapter 5: Implemention and Testing **10**

6 Chapter 6: Results and Discussion **10**

7 Chapter 7: Conclusions **10**
7.1 subsection 10
7.1.1 subsubsection 10

8 image **10**

1 Chapter 1: Introduction

1.1 Importance of Testing

How severe can the consequences be from an error in a piece of software? In 1983 a bug in a piece of software nearly started World War Three.

During the Cold War, tensions between the US and Soviet Russia were extremely high. A Soviet early warning system had detected the launch of five ballistic missiles from the US. The only reason that Soviet Russia did not retaliate, starting World War Three, was the fact that Lt Col Stanislav Petrov had a "...funny feeling in my gut"[1] and concluded that if the US was launching a full scale attack they would launch more than five missiles. The error in the system was discovered to be a bug in part of the software that distinguished false missiles from satellites picking up the reflection of sunlight from the top of clouds.[1] If the bug in the code has falsey detected more missiles the world could be a very differn't place today.

Software disasters are caused by three main reasons. Poor software project management, poor risk assessment and poor development and testing practices.[2] Software testing is therefore extremely prevelent and has upmost importance as it can prevent World War and save companies a lot of money.

1.2 Benefits of Testing and Automation

Inadequate software testing infastructure was estimated to cost the US economy $59.5 billion a year.[3] It was also estimated that the potentail cost reduction from feasbile infastructure would be $22.2 billion a year. [3] Due to software disasters and the vast amount of money that can be saved and also avoid incurring additonal costs, people have become more aware of the importance of testing. The benefits of the reduced costs comes from ihe increased reliabilty and quality of the product produced when software testing is implemented.

"In a typical programming project approximately 50 percent of the elapsed time and more than 50 percent of the total cost were expended in testing the program or system being developed"[4]. Software testing saves you alot of money but also costs alot of money. By automating part or the whole process the costs can be reduced while still obtaining all of the benefits.

1.3 Ruby on Rails

Ruby on Rails as a framework

model as mvc

Rspec to test m

1.4 Project Aims

The overall aim of this project is to reduce the cost of developing Ruby on Rails applications. The reduction in costs comes from the time saved by automatically generating test cases for the model component of the application. The develop will input a formal specification of thier database into a system from which they can download seperate files containing a test suite for each table they have defined. The files are seperated in keeping with standard Ruby on Rails practices. Once the file is downloaded it can be inserted into the application and be available to run immediatley. This project should reduce the errors and bugs in Ruby on Rails applications via the feedback from the tests generated, improving the reliablity and quality of Ruby on Rails projects via the model component.

1.5 Project Limitations

Rails and its do more with less, inline with automated testing..

The aims of the project are to:

1. Reduce the amount of time it takes for a User to produce tests for model validation
2. Produce tests that are of high readable quality
3. Produce tests that fully test properties specified
4. The process should be hassle free

Challenges that the project faces are

1. Identifying the minimum information required to produce tests
2. Creating a process that is hassle free
3. Natural language in tests that is appears human written
4. Generating the tests in an acceptable time frame
5. Building an efficent database structure for the information

2 Chapter 2: Literature Review and Research

2.1 Tools to use

add a bit about bootstrap

The main requirements for choosing what tools to use to build the application are:

1. Allowing a User to enter information onto a database
2. Allowing a User to view information on the database
3. A system that can process the information and generate Test cases

An MVC web application fits all these criteria while providing additional benefits. MVC, Model-View-Controller is an architectural pattern that seperates an application into three interconnected parts. The separation allows for responsibilities to be allocated independently to each component, seperating the logic from the user. The model is responsible for the data of the application and the rules and logic used to create and update the information. The view is responsible for displaying the data and possible interactions with the system to the user. The controller is responsible for controlling the flow of the system, accepting user input and converting it into commands for the model and view.

An example of the components interacting would be creating an entry to a database. The view would be responsible for displaying the form in which to fill in. On submission the controller will process the information, ensure only persmissable information is submitted and enter additional information, then send it to the model. The model would verify the structure of the information, correct fields are present and cohere with its rules. The model will then notify the controller if the submission was successful or not and the controller will update the view to reflect the status.

The seperation means that all user interaction with the database has to be verified by the controller. This provides a high level of security as each action is controlled and the internal structure and representation of the information within the database is hidden. Simultaneous development is also possible due to the seperation of the components, work on the front and back end concurrently. Although I will not be able to get the full benefit of this as I am developing the project solo, it will allow me to shift focus as components do not need to finished before switching to another, giving greater flexibility in development.

High levels of cohesion are inherited automatically from the architecture with the grouping of logically similar elements, this makes the code easier to read and creates a more natural flow within the source code. There are however some drawbacks to MVC architecture, they are inheritly more complex due to the seperation and the framework must be learnt in addition to the programming language it is in, which there can be multiple languages between the components.

This steep learning curve could mean a large initail investment into a team learning a new framework along with its languages.

MVC web application frameworks have become extremely popular and are behind some of the most used and powerful websites. Django an MTV, follows MVC architecture but its creators decided to rename the components [5] to better suit them, is behind the two most visted websites in the world Google and YouTube[6][7]. Ruby on Rails another MVC is behind Twitter, Airbnb and Soundcloud.[8] MVC frameworks are known for their scalablity, being suitable for the smallest to the largest projects. However FaceBook decided that its scalablity had reached its limit, that adding new features made the code exponentailly more complex.[9] My project will be no where near the scale of facebooks sourcecode so I do not need to worry about reaching the end of its scalibilty.

The chosen MVC to construct the project in is Ruby on Rails. I have done previous projects in both Ruby and the Rails framework, also the novelty of contstructing the software in the software that its output will test.

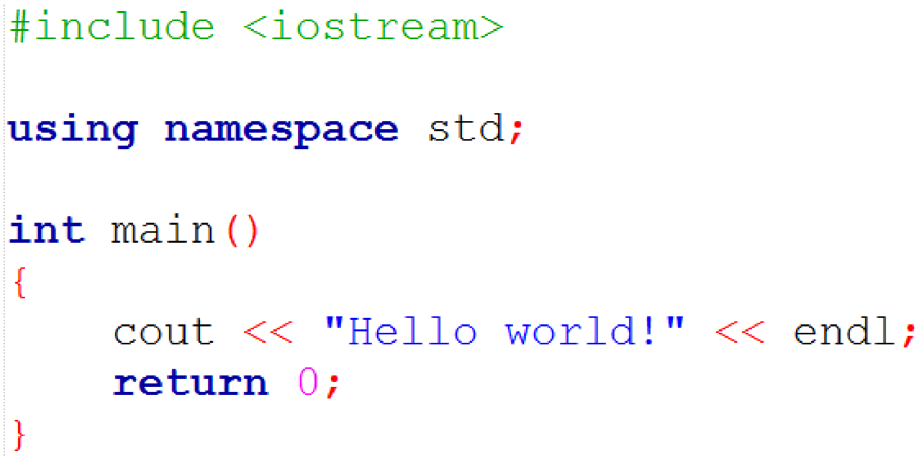
Ruby was selected as the primary programming language by default as it is the language that runs Ruby On Rails. Ruby is a dynamic, multi-paradigm programming language. The paradigms consist of Object-oriented, Imperative, Functional and Reflective making it a very powerful and versatile language. This combination is from its founder Yukihiro Matsumoto who was influenced by Perl, Smalltalk, Eiffel, Ada, and Lisp.

Rubys primary design goal was to make a language that he himself enjoyed using, by minimizing programmer work and possible confusion(Ruby Wiki). Achieved with a focus on human interaction, how programmers code and design applications as opposed to focusing on how the code will run on machines. And also following the principle of least astonishment, where the behaviour of the language minimizes confusion for experienced users.

The above two images show C++ 1 and Ruby 2 printing Hello world to the console. The comparison between the two languages highlights the efficiency and simplicity of the Ruby language.Ruby on Rails projects are commonly worked on by a group of people and in multiple languages, therefore the simplistic syntax gives greater clarity and understandability to programmers who are lesser experienced in Ruby.[10]

Ruby is open source, free and redistributable with a vast range of existing code from both Ruby and its large community. Primarily consisting of Gems, code packages that can be installed and supported into a project easily and with minimal effort via RubyGems, and frameworks, such as Ruby on Rails. Making it very popular for education and business. Following the DRY Don't Repeat Yourself principle in a very effective and efficient manner.[10]

Ruby was ranked ninth on TIOBE index[**?**] and has become a very popular and respected language relative to its age among the other languages on the index.

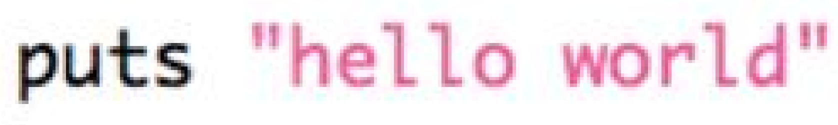


```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Figure 1: C++ print Hello world to console



```
puts "hello world"
```

Figure 2: Ruby print Hello world to console

No alternatives could be considered due to the dependency of Ruby on Rails on Ruby, however Ruby is a very strong and durable language so it does not detract from the overall project.

paragrpah about rails, more its specifics

2.2 Testing and Automation

The European Space Agency spent ten years and $7 billion designing and constructing the Ariane 5, a rocket that can launch multiple satellites into orbit from a single launch. Thirty nine seconds into its maiden voyage it exploded, destroying the Ariane 5 and its contents of four uninsured, extremley expensive scientific satellites. The explosion was caused by its own self-destruct sequence which was triggered automatically as the boosters were being torn away by aerodynamic forces. The extreme aerodynamic forces were caused by the rocket trying to recorrect its course due to flight data provided the guidance system. The guidance system had crashed and shutdown, along with its backup, the flight data provided that caused the rocket to readjust its course was actually a diagnostic error message. The cause of the shutdown was the guidance system trying to convert the sideways velocity of the rocket from 64-bit to 16-bit where an overflow occured causing the system to shutdown. To make matters worse, the programmers were aware that it could overflow but assumed that particular variable would never be large enough as it was used to prepare for launch and not in flight. However it was decided the system should run into the first forty seconds of flight, incase of a brief hold in the launch countdown, to make restarting the system easier. A known flaw in a system, that could of been handled, ended up causing the chain of events that led to the rocket exploding.[11]

Software testing is an investigation into a piece of software that provides information during development and maintenance. A process or series of process's are carried out that are designed to make sure computer code does what it designed to do and is absent of unintended behaviour. [4]The information retrived from the proccess can be used to track the progress during development against acceptance criteria. Errors and bugs detected within the code of the are immediatley known and can be handled, providing a smoother and more consistent development and maintenance flow. Software testing provides a more reliable and higher quality product when used as part of the development process.

Testing can not guarantee that a program or peice of code is without errors, therefor completing testing is impossible. This is why the design of tests is vital to the integrity of the testing, making the tests as complete as possible. Given the constraints on time and cost, effective testing is simply "What subset of all possible test cases has the highest probability of detecting the most errors?"[4]. Tests are designed using information about the program along with its intended behaviour. In a given environment, with proper determined input, there is an expected behaviour/output. If the code undertest does not display the desired outcome it is said to of failed the test. Design of tests is important and to design a test information is required, the information is sourced in two main ways Black Box and White Box.

Black box also known as Functional Testing is the technique of creating test cases with information from the software requirements and or its design specification. The software entity under test is treated as a black box where proper inputs are fed in and outputs are observed with no concern to the actual structure of the code. Black box testing can detect some behaviours that white box cannot, such as absent behaviours that are in the requirements but not coded in.[12][13]

White box testing, structural testing uses information from the physical code to produce test cases. The derivation of information makes the test more focused on the actual implementation of the code rather than its specified behaviour. Test cases tend to be at a much finer grain than black box testing individual methods. Tests are designed to execute a particular behaviour within the program, such as testing how it handles a binary overflow.[12][13]

As Ruby on Rails design environment can vary drastically between projects due to the flexibility of its framwork and use of Gems I will only consider Black Box testing techniues when I come to designing the project. This will deliver a product that will be usable to a wider audience as it is dependent upon on specification for which I can define. Interpreting multiple languages and being flexible enough to be useful is out of the scope of this project.

A 'test case' will test a very specific behaviour of a program. A collection of test cases is a 'test suite', representing that a certain section of the system has a specificied set of behavoiurs. A relevant example would be for a table in a database. The test suite would represent if the table has the desired validations in place and would consist of test cases that tested each specific behaviour in isolation. The tests would be run as a set to confirm the table has the desired behaviour, while being in the case of undesired behaviour being to specify which test case, therefore highlight the exact error in the code. Therefor to automate test generation, test cases need to be automatically generated with being able to identify the necessary cases to complete a set.

Software testing is necessary and very costly. "In a typical programming project approximately 50 percent of the elapsed time and more than 50 percent of the total cost were expended in testing the program or system being developed"[4]. Reducing the costs, both time and monetary are the main motivations for Automated Testing. Another overlooked and unappreciated benefit of automating testing is test case generation is one the most intellectully demanding and critical challenges in software testing.[14] By automating this process it allows the programmers to dedicate not only more time but more of thier effort to other areas, also in some cases it is harder to create a test case but easy to verify a generated test case is correct. A whole systems tests do not have to be automatically generated to reduce costs and benefit.

Automated functional testing follows the same method as manual generation of test cases. The information used to derive test cases is a functional specification: description of intented program behaviour distinct from the program itself. In manual this can be formal or informal, however in automated it is formal so that it can be interpreted by the system that will generate the test cases. The possible behaviours of the program derived from the formal specification are divided into test suites then again into test cases. For a database, tables would be divided into test suites, then as before the properties of the fields would be the test cases. The systematic nature can help avoid missed test cases and provide more consistent coverage. When automated there is more creativitivy and design put into the functional specification as the test designer is usually limited to a choice of test selection criteria.[13]

As observed with automated functional testing, a system is used to interpret and understand the same information that a human would use to produce test cases, effectively replacing the human user. Automated white box testing tends to be subsequently more complex, as it has to understand and interpret human written code. One method that has recieved alot of attention from researchers is Symbollic Execution, where symbollic values are used to instead of concrete values program inputs, the programs variables are then be described by the symbolic expressions of those inputs. The state of the program includes the symbolic values of program variables, a program counter and the path contraint on symbolic values: a boolean formula over the symbolic values input. Using this method it can explore all possible path divergences through a system and identify stop points, where the path ends. The major problem with automated white box testing is identifying if a behaviour, stop point or a specific divergence in Symbollic execution, is desired or not. This problem is known as the Oracle problem, as desired behavoiur of code is contained within its specfication and design, not its implementation. Therefor automated test generation must always include some level of user input. Another relevant challenge Symbolic Execution is developing a system that devloping system that can cover multiple languages at once is very complex and producing a system can produce feasible output can be impossible due to the path divergence problem, where either a user has to specify so many models it isn't automated wnough to benefit or it doesnt find a signifcant amount of feasible program paths.

2.3 Testing in Rails

various methods consistent models

test unit rspec

factory girl

rspec examples

suite example

2.4 wrapup

conclusion: research influenced project only model, black box, rspec, produce written tests etc etc

2.5 Evaluation

how testing is evaluated, code coverage bugs found etc

how other people tested, difficulties

maniuplation

dog feed

compare with existing

compare code coverage, how much of theres did it do

limited so not going to cover all, but does it save a signifacant amount of time

3 Chapter 3: Requirements and Analysis

Filler text, filler text, filler text, filler text, filler text, filler text, filler text, filler text.

4 Chapter 4: Design

front end - bootstrap - clean - easy to use

information required to generate test cases

how information should be entered

how information should be presented/edited

database design to accomadate the information

flow of program code to evaluate efficently

templates for rspec that can be filled in

generate button

number generation

string generation

output should be a file that is same as user created

5 Chapter 5: Implemention and Testing

Filler text, filler text, filler text, filler text, filler text, filler text, filler text, filler text.

6 Chapter 6: Results and Discussion

Filler text, filler text, filler text, filler text, filler text, filler text, filler text, filler text.

7 Chapter 7: Conclusions

Filler text, filler text, filler text, filler text, filler text, filler text, filler text, filler text. Hello World!

7.1 subsection

Structuring a document is easy!! [15]

7.1.1 subsubsection

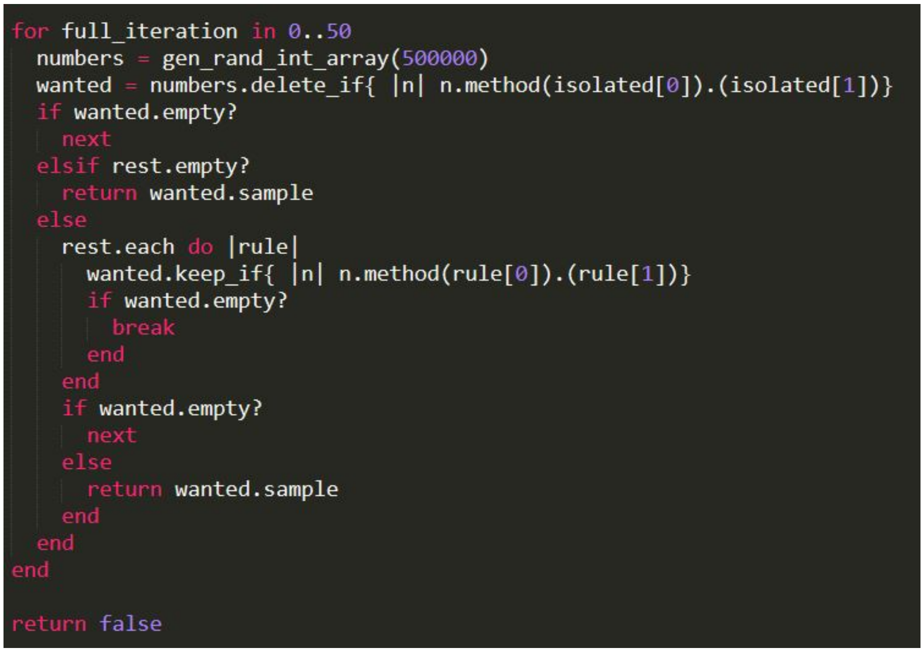
p1 It's a me, Mario[1].

sp1 Wwoohooo

8 image

Figure 3 shows some savage code

[1] [15]



```
for full_iteration in 0..50
  numbers = gen_rand_int_array(500000)
  wanted = numbers.delete_if{ |n| n.method(isolated[0]).(isolated[1])}
  if wanted.empty?
    next
  elsif rest.empty?
    return wanted.sample
  else
    rest.each do |rule|
      wanted.keep_if{ |n| n.method(rule[0]).(rule[1])}
      if wanted.empty?
        break
      end
    end
    if wanted.empty?
      next
    else
      return wanted.sample
    end
  end
end

return false
```

Figure 3: caption for image, shown below image

Table 1: Caption for the table.

Some	actual	content
prettifies	the	content
as	well	as
using	the	booktabs package

References

[1] C. Barker, “The top 10 it disasters of all time,” 2007.

[2] P. A. McQuaid, “Software disastersunderstanding the past, to improve the future,” *Journal of Software: Evolution and Process*, vol. 24, no. 5, pp. 459–470, 2012.

[3] RTI, “The economic impacts of inadequate infrastructure for software testing,” in *Planning Report 02-3 The Economic Impacts of Inadequate Infrastructure for Software Testing*, p. 309, NIST, 2002.

[4] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[5] Django, “Django faq,” 2017.

[6] SHUUP, "25 of the most popular python and django websites," 2015.

[7] Alexa, "The top 500 websites," 2017.

[8] Coderfactory, "Top 15 sites built with ruby on rails," 2017.

[9] Infoq, "Facebook: Mvc does not scale, use flux instead," 2014.

[10] Ruby, "About ruby," 2016.

[11] J. Gliek, "A bug and a crash," 1996.

[12] S. Nidhra and J. Dondeti, "Blackbox and whitebox testing techniques-a literature review," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.

[13] M. Young, *Software Testing and Analysis: Process, Principles, and Techniques.* Wiley India Pvt. Limited, 2008.

[14] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[15] J. P. Near and D. Jackson, "Rubicon: bounded verification of web applications," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, p. 60, ACM, 2012.