Unit Testing

Managing Implementation Complexity

- Individual parts should be verified before being integrated with other parts
- Integrated subsystems should also be verified
- If adding a new part breaks the system, the problem must be related to the recently added part
- Track down the problem and fix it
- This ultimately leads to a complete system that works

2 Approaches to Programming

- Approach #1
  - "I wrote ALL of the code, but when I tried to compile and run it, nothing seemed to work!"

- Approach #2
  - Write a little code (e.g., a method or small class)
  - Test it
  - Write a little more code
  - Test it
  - Integrate the two verified pieces of code
  - Test it
  - …

Unit Testing

- Large programs consist of many smaller pieces
  - Classes, methods, packages, etc.
- "Unit" is a generic term for these smaller pieces
- Three important types of software testing are:
  - Unit Testing (test units in isolation)
  - Integration Testing (test integrated units)
  - System Testing (test entire system that is fully integrated)
- Unit Testing is done to test the smaller pieces in isolation before they are combined with other pieces
  - Usually done by the developers who write the code

What Unit Tests Do

- Unit tests create objects, call methods, and verify that the returned results are correct
- Actual results vs. Expected results
- Unit tests should be automated so they can be run frequently (many times a day) to ensure that changes, additions, bug fixes, etc. have not broken the code
  - Regression testing
- Notifies you when changes have introduced bugs, and helps to avoid destabilizing the system

Test Driver Program

- The tests are run by a "test driver", which is a program that just runs all of the unit test cases
- It must be easy to add new tests to the test driver
- After running the test cases, the test driver either tells you that everything worked, or gives you a list of tests that failed
- Little or no manual labor required to run tests and check the results

JUnit Testing Design

- Write a separate test method for each test
  - Marked with @Test annotation
- Set up method(s) may be executed before each test method
  - Marked with @BeforeEach or @BeforeAll
- Tear down method(s) may executed after each test
  - Marked with @AfterEach or @AfterAll
- Use JUnit `Assertions.assert*()` methods to implement test cases
- Failures reported in various ways, depending on language and tool (command-line, GUI, IDE integrated)
- Example:
  - WordExtractor.java
  - WordExtractorTest.java

Running Junit Tests from IDE's

- To run a single test class, in the "Project" tool window right-click on a test class name, and select "Run Tests" or "Debug Tests"

- To run all of your unit tests, right-click on the "test/java" folder, and select "Run All Tests" or "Debug All Tests"

Running Unit Tests from The Command-Line

- Write a test driver class whose "main" method invokes the org.junit.runner.JUnitCore class to run your unit tests

- Run your test driver program from the command-line:

  java –cp build\classes\main;build\classes\test;libs\junit-jupiter-api-5.5.1.jar;libs\junit-platform-console-1.5.1.jar;libs\sqlite-jdbc-3.25.2.jar TestDriver

JUnit 5 Unit Testing Framework

- JUnit 5 Documentation
- Use JUnit 5 annotations to mark test methods

Annotation	Description
@Test public void method()	The annotation @Test identifies that a method is a test method.
@BeforeEach public void method()	Will execute the method before each test. Can prepare the test environment (e.g. read input data, initialize the class).
@AfterEach public void method()	Will execute the method after each test. Can cleanup the test environment (e.g. delete temporary data, restore defaults).

JUnit 5 Unit Testing Framework

Annotation	Description
@BeforeAll public void method()	Will execute the method once, before the start of all tests. Can be used to perform time intensive activities, for example to connect to a database.
@AfterAll public void method()	Will execute the method once, after all tests have finished. Can be used to perform clean-up activities, for example to disconnect from a database.
@Timeout(5)	Fails if the method takes longer than 5 seconds.
@Timeout(value = 100, unit = TimeUnit.MILLISECONDS)	Fails if the method takes longer than 100 milliseconds

Adding the JUnit Library to Your Project

- Maven

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.5.1</version>
    <scope>test</scope>
</dependency>
```

- Gradle (build.gradle file)

```
testCompile group: 'org.junit.jupiter', name:
'junit-jupiter-api', version: '5.5.1'
```

Database Unit Tests

- When writing unit tests for your database code, there are additional things to think about
- Put database driver JAR file on the class path
- Each unit test should start with a pristine database so prior tests have no effect
  - Can re-create tables before each test
  - Or, you can "rollback" the effects of each test so they are undone and don't affect later tests

Code Quality

Best Practices in Coding

- Some basic issues in every development team
  - fundamental discipline issues in coding
  - number of years of exp. does not match delivery
  - by the time they actually become good coders, they are promoted to leads
  - 70% of the whole employees, belong 0-4 years of experience

- In India, senior developer means 3-5 years of exp, in USA, senior developer means 6-10 years of exp – There is a huge gap in expectation and quality

- Coding is not rocket science

- The rework time in coding is the actual killer
- 20-25% of the time is spent in reworking in software – that means, fix the mistakes done

Coding Practices - Introduction

- In a mobile phone, can you insert sim card wrongly? You cannot. Because the sim card design is like that. So the highest level maturity is not to give an opportunity to make mistakes

- As a kid, you were asked and helped to brush the teeth. It was tough to brush teeth.
- One fine morning, you start brushing teeth by yourselves.
- Then it has become a habit. You need not be told to do brushing by anyone.

Commenting

- Commenting is necessary for maintaining any program
- Without commenting, the program is not complete
- Do not assume that the other person who reads this program will understand it clearly
- If commenting is not done at the beginning, it is forgotten.

- Program - a physical file – you need to comment the following at the top.
  - at the top of the program - header comment
  - what the program achieves
  - who coded on what day

- Classes
  - Comment at the beginning of the class about what the class does
  - class Employee
  - // this class is used to manage all information
  - // about employees and retrieve specific data

- Methods within classes
  - what this method/function/procedure does
  - what are the parameters and their purpose
  - what are the return values

- Comment before every loop.

- Comment before every file or database operation

Commenting

- If you are team leader, if your team member comes with a code, that does not have these comments – what will you do?
- You take over a program for maintenance from another person, and that program has no comments – what will you do?
- Your company hands over a project to client. Programs do not have comments. What the client will say?

Naming Conventions

- Readability improves understanding; that further improves maintainability
- If programs are not consistent, maintenance is tough
- If 20 developers work in a project and each one names functions in the way he/she wants, will the client approve the same?
- We must have a document on naming conventions
- Every document must be named properly and it must be stored in a proper folder.

Naming Conventions

- You need to have a convention for naming
  - Programs
  - Classes
  - Methods
  - Variables
  - Labels in programs
  - Reusable library functions
- There are established methods like Pascal casing, camel casing etc.
- Usually you will tend to use a mixture of uppercase, lowercase letters, underscore etc
- The ultimate aim is to achieve consistency across programs

Avoid Hard coding

- Placing a number or quoted text inside active code is hard coding
- This is deadly as the program restricts itself to this hard coded value
- To make a change, you need to change code, recompile and redeploy
- Usual way to avoid hard coding is to use constants at the top of a program; here also, you need to edit the program, but the change is in one place
- Other way is to put all configurable values in a property file. Every program must read the name value pair from the file and later use them in the code
- Eg. MAX_LENGTH 150, PORT 8097

Modularity of code

- Modularity is important for easy maintenance
- Do not write lengthy methods or procedures
- We can split the modules based on
  - A functional operation as per requirements
  - A piece of code is used in more than one place
  - Logical breakdown of events in the functionality
- Modularity must be decided right at the design level itself
- Modular programs are easy to debug
- Fixes done on modular programs are easy to isolate from other regression effects

Examine the loops

- A loop is - repeat some logic for specific number of times or as long as a condition is true/false
- If this is not checked, it can run forever, infinitely and can bring down the server
- Loop Sample
  - gateway 1 // variables, flags
  - xyz = 10
  - loop starts here
  - ...gateway 2 // variables, flags, counters
  - ..
  - logic
  - ... gateway 3
  - loop ends here
  - gateway 4
- Gateways are the check posts where we must watch the value of the variables, loop counters and loop flags
- Ensure proper resetting of flags and counters and check their values and gateways of the loops
- Never allocate memory inside a loop
- Never instantiate a class inside a loop; if needed, close it within the loop
- Usually companies do not suggest more than 3 levels in the loops

Exceptions

- Issues may be caused by programmer
- Issues may be caused by system
- Most of the languages allow try-catch or on-error-goto exceptions

- Exception is also an error condition
  - we do not know when it will happen

- 1. Any file operations – handle exceptions, because
  - file may not exist
  - file is already opened by someone
  - file does not have privilege
  - file is already full

- 2. Any database operation – handle exceptions, because
  - database you may not have rights
  - database is down
  - connections exhausted

- 3. Memory
  - low memory exception
  - pointers - writing in privileged memory
  - array boundary breach

- 4. Any external evices
- your program accesses webcam or printer

Performance

- We need to monitor cpu, memory, network
- Memory is directly related to variables and object – do not declare huge arrays or objects
- Cpu is consumed more when you deal with db or files or devices
- Any database operations, open late and close the connection early.
- Usage of indexes in queries must be examined
- Any column used in where condition of db query must have index on it
- Any memory allocated must be freed – else the program will shut down after some time
- Any object instantiated must be released – else system will be depleted of memory and hence performance will come down

Program Logic

- Usually the logic design will be provided to the developer
- If it is not provided, take 30 minutes and write the logic of the program in English first
- Do not start coding right away. You will make so many assumptions and it will spoil the show
- Get the logic approved by the team lead and then start coding
- Developers usually feel that this takes time; but it reduces the time effectively while coding and reworking
- Since developers are not used to documenting, they feel it is not their job. Hence they miss a lot of finer points
- If you write the logic first, you will get a lot of clarifications at that time itself and hence the code will come out clean

Code Quality Tools

SonarQube

- Web-based tool to measure and analyze the quality of source code
- Usable for any GN4-3 participant
- Many integration options
- Quality Gate
- Measures
  - Reliability, Security, Maintainability (with simple “A” to “E” rating)
  - Test Coverage, Duplications
  - Size, Complexity
- Quality Profiles

SonarQube -How

- Add your project to SonarQube
- Prepare SonarQube scanner
- Let SonarQube assess your project
- Analyze the results

DevOps

What exactly is DevOps?

Means different things to different people

DevOps aims to bring applications to market rapidly through:

- Cross-functional empowered teams (Business, Dev, Ops, QA) with full lifecycle responsibility for delivering a service
- Rapid code release cycles
  - But large variation in release frequency
- Agile Development
  - Trunk-based with Feature Flags
- Service Orientated Architecture on a cloud-based infrastructure
- Tool-chain automation
  - Continuous Integration/Continuous Testing/Continuous Delivery

Security is perceived an inhibitor

- Penetration test based release gate is too slow

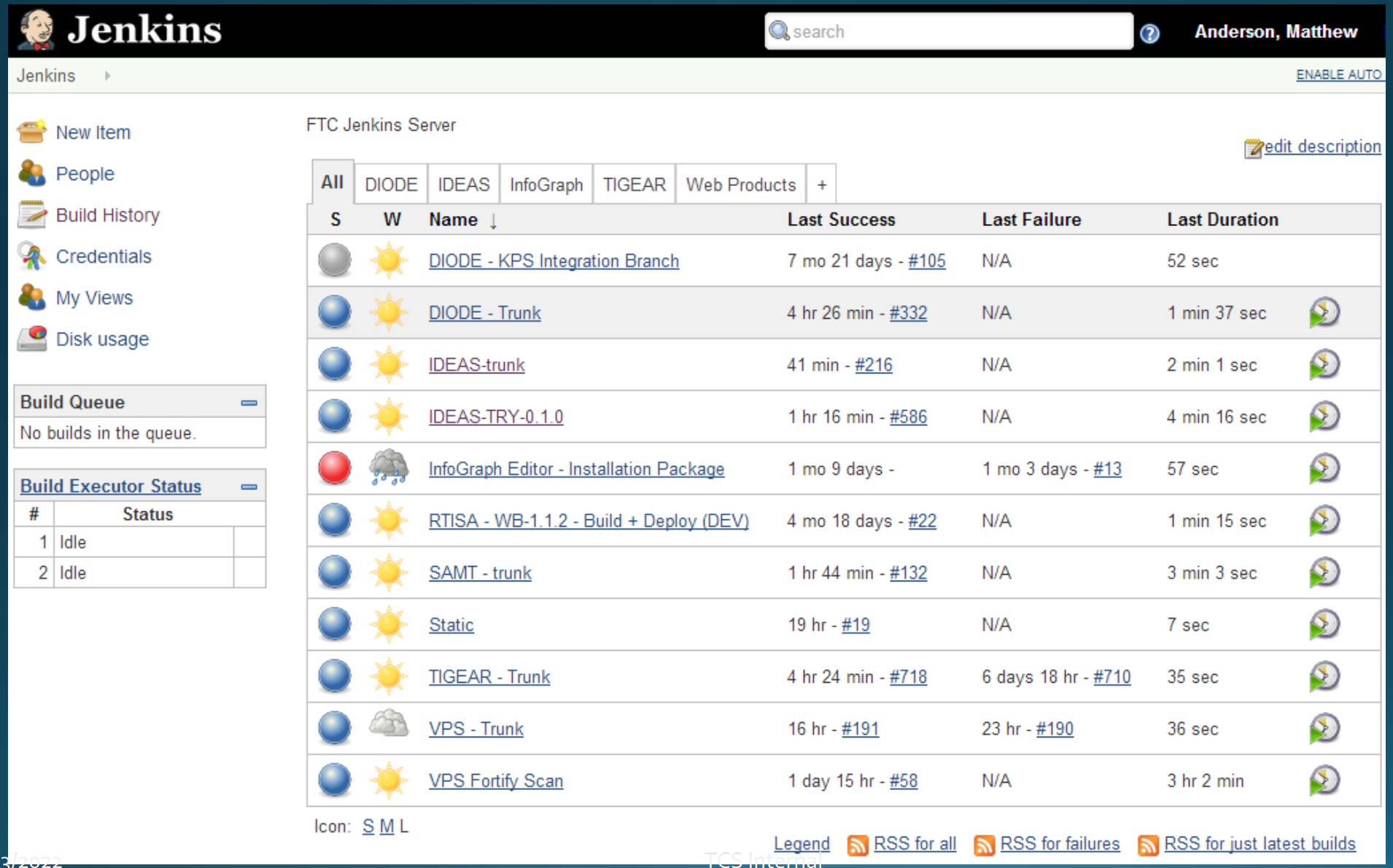
Jenkins

- Open Source Continuous Integration Server
- Monitors executions of repeated jobs
- Used primarily for continuously building and testing software projects, i.e. “Continuous Integration” or “CI”
- Previously known as “Hudson”
- http://jenkins-ci.org

Why Use Jenkins

- Immediate feedback on broken builds
- No building from dev machines and manually copying files
- Automated deployment
- Automated test execution and feedback

Jenkins- Dashboard



Jenkins- Project View

Jenkins

search

Anderson, Matthew

Jenkins ▸ IDEAS-TRY-0.1.0 ▸

ENABLE AU

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- Subversion Polling Log

Build History (trend)

#587	Oct 21, 2014 10:54:49 AM	3 MB
#586	Oct 21, 2014 9:24:51 AM	3 MB
#585	Oct 20, 2014 4:24:49 PM	3 MB
#584	Oct 20, 2014 3:24:48 PM	3 MB
#583	Oct 20, 2014 10:09:48 AM	3 MB
#582	Oct 20, 2014 9:09:46 AM	3 MB
#581	Oct 17, 2014 4:26:52 PM	3 MB
#580	Oct 17, 2014 2:24:42 PM	3 MB
#579	Oct 17, 2014 10:24:39 AM	3 MB
#578	Oct 16, 2014 9:39:41 PM	3 MB
#577	Oct 16, 2014 5:09:39 PM	3 MB

RSS for all RSS for failures

Project IDEAS-TRY-0.1.0

add description

Disable Project

Project disk usage information + trend graph

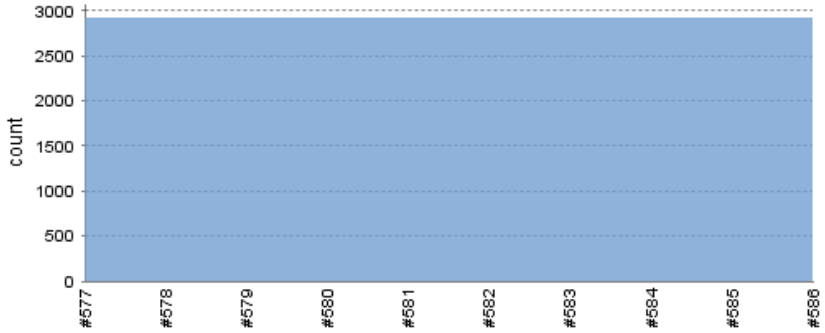
Workspace

Disk Usage: Workspace 1 GB (On slaves -, Non slave workspaces 1 GB), Builds 30 MB (Locked -)

Recent Changes

Latest Test Result (no failures)

Test Result Trend



(just show failures) enlarge

Permalinks

- Last build (#587), 3 min 7 sec ago
- Last stable build (#586), 1 hr 33 min ago
- Last successful build (#586), 1 hr 33 min ago

Security Assessments - SAST

- **White box security testing** The tester has access to the underlying framework, design, and implementation. The application is tested from the inside out. This type of testing represents the developer approach.
- **Requires source code** SAST doesn't require a deployed application. It analyzes the sources code or binary without executing the application.
- **Finds vulnerabilities earlier in the SDLC** The scan can be executed as soon as code is deemed feature-complete.
- **Less expensive to fix vulnerabilities** Since vulnerabilities are found earlier in the SDLC, it's easier and faster to remediate them. Findings can often be fixed before the code enters the QA cycle.
- **Can't discover run-time and environment-related issues** Since the tool scans static code, it can't discover run-time vulnerabilities.
- **Typically supports all kinds of software** Examples include web applications, web services, and thick clients.

Security Assessments - DAST

- **Black box security testing** The tester has no knowledge of the technologies or frameworks that the application is built on. The application is tested from the outside in. This type of testing represents the hacker approach.
- **Requires a running application** DAST doesn't require source code or binaries. It analyzes by executing the application.
- **Finds vulnerabilities toward the end of the SDLC** Vulnerabilities can be discovered after the development cycle is complete.
- **More expensive to fix vulnerabilities** Since vulnerabilities are found toward the end of the SDLC, remediation often gets pushed into the next cycle. Critical vulnerabilities may be fixed as an emergency release.
- **Can discover run-time and environment-related issues** Since the tool uses dynamic analysis on an application, it is able to find run-time vulnerabilities.
- **Typically scans only apps like web applications and web services** DAST is not useful for other types of software.