PENERAPAN TEST DRIVEN DEVELOPMENT PADA APLIKASI ANDROID SISTEM INFORMASI PETANI HUTAN (STUDI KASUS DINAS KEHUTANAN PROVINSI LAMPUNG)

(Skripsi)

Oleh

THEOFANI HATI KUSUMAWARDANI
NPM 2255061004

FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG
BANDAR LAMPUNG
2026

PENERAPAN TEST DRIVEN DEVELOPMENT PADA APLIKASI ANDROID SISTEM INFORMASI PETANI HUTAN (STUDI KASUS DINAS KEHUTANAN PROVINSI LAMPUNG)

Oleh

THEOFANI HATI KUSUMAWARDANI

Skripsi

Sebagai Salah Satu Syarat untuk Mencapai Gelar
SARJANA TEKNIK

Pada

Jurusan Teknik Elektro
Fakultas Tenik Universitas Lampung

FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG
BANDAR LAMPUNG
2026

ABSTRAK

PENERAPAN TEST DRIVEN DEVELOPMENT PADA APLIKASI ANDROID SISTEM INFORMASI PETANI HUTAN (STUDI KASUS DINAS KEHUTANAN PROVINSI LAMPUNG)

Oleh

THEOFANI HATI KUSUMAWARDANI

Dinas Kehutanan Provinsi Lampung menghadapi tantangan aksesibilitas dalam pelaporan data Nilai Transaksi Ekonomi (NTE) akibat keterbatasan konektivitas internet dan keterbatasan perangkat yang dimiliki di lapangan. Pengembangan aplikasi dengan fungsionalitas *offline-first* diperlukan dalam menghadapi tantangan tersebut, namun manajemen data *offline* dan logika sinkronisasi memiliki tingkat kompleksitas yang tinggi, sehingga berisiko meningkatkan *bug* dan menurunkan keterpeliharaan (*maintainability*) sistem di masa depan. Penelitian ini bertujuan membangun aplikasi *offline-first* android Sitanihut dengan penerapan metode *Personal Extreme Programming* (PXP) dan pendekatan *Test Driven Development* (TDD). Hasil pengembangan selama enam iterasi menunjukkan bahwa penerapan TDD dengan total 563 *unit test* menghasilkan *code coverage* sebesar 95,53%. Evaluasi kompleksitas logika menghasilkan skor McCabe *Cyclomatic Complexity* sebesar 2,36 dan kompleksitas pemahaman kode *Cognitive Complexity* sebesar 1,77, membuktikan rendahnya kompleksitas kode yang dibangun. Pengujian akhir melalui *User Acceptance Test* (UAT) menghasilkan skor sebesar 100% atau dapat diterima dari tiga kelompok pengguna.

Kata kunci: *Android, Offline-first, Personal Extreme Programming, Test Driven Development, Code Coverage, Cyclomatic Complexity, Cognitive Complexity, User Acceptance Test*

ABSTRACT

TEST DRIVEN DEVELOPMENT IMPLEMENTATION ON MOBILE ANDROID APPLICATION FOR SISTEM INFORMASI PETANI HUTAN (CASE STUDY: DINAS KEHUTANAN PROVINSI LAMPUNG)

By

THEOFANI HATI KUSUMAWARDANI

Dinas Kehutanan Provinsi Lampung *faces accessibility challenges in reporting Economic Transaction Value (NTE) data due to limited internet connectivity and limited devices available in the field. Application development with offline-first functionality is necessary to address these challenges, but offline data management and synchronization logic have a high level of complexity, thus risking increasing bugs and reducing system maintainability in the future. This study aims to build an offline-first android application Sitanihut by implementing the Personal Extreme Programming (PXP) method and the Test Driven Development (TDD) approach. The development results for six iterations show that the application of TDD with a total of 563 unit tests resulted in a code coverage of 95.53%. The evaluation of logic complexity resulted in a McCabe Cyclomatic Complexity score of 2.36 and a Cognitive Complexity code understanding complexity of 1.77, proving the low complexity of the code built. Final testing through the User Acceptance Test (UAT) resulted in a score of 100% or acceptable from three user groups.*

Keywords: *Android, Offline-first, Personal Extreme Programming, Test Driven Development, Code Coverage, Cyclomatic Complexity, Cognitive Complexity, User Acceptance Test*

Judul Skripsi	: PENERAPAN TEST DRIVEN DEVELOPMENT PADA APLIKASI ANDROID SISTEM INFORMASI PETANI HUTAN (STUDI KASUS DINAS KEHUTANAN PROVINSI LAMPUNG)
Nama Mahasiswa	: Theofani Hati Kusumawardani
Nomor Pokok Mahasiswa	: 2255061004
Program Studi	: Teknik Informatika
Fakultas	: Teknik

MENYETUJUI

1. Dosen Pembimbing

Pembimbing Utama	Pembimbing Pendamping
Wahyu Eko S, S.T., M.Sc.	Sony Ferbangkara, S.T.,M.T
NIP 19741201 200112 1 001	NIP 19820217202321 1 014

2. Mengetahui

Ketua Jurusan Teknik Elektro	Ketua Program Studi Teknik Informatika
Herlinawati, S.T., M.T	Yessi Mulyani, S.T., M.T
NIP 197103141999032001	NIP 197312262000122001

MENGESAHKAN

1. Tim Penguji

Ketua : Wahyu Eko S, S.T., M.Sc.

Sekretaris : Sony Ferbangkara, S.T.,M.T

Penguji : Mona Arif Muda, S.T., M.T

2. Dekan Fakultas Teknik

Dr. H. Ahmad Herison, S.T.,M.T.
NIP 196910302000031001

Tanggal Lulus Ujian Skripsi: 21 April 2026

SURAT PERNYATAAN

Dengan ini saya menyatakan bahwa dalam skripsi ini tidak terdappat karya yang pernah dilakukan orang lain dan sepanjang pengetahuan saya tidak terdapat atas diterbitkannya oleh orang lain, kecuali secara tertulis diacu dalam naskah ini sebagaimana yang disebutkan di daftar pustaka. Selain itu, saya menyatakan pula bahwa skripsi ini dibuat oleh saya sendiri.

Apabila pernyataan saya terbukti tidak dapat dipertanggungjawabkan kebenarannya, saya bersedia dikenai sanksi akademik sesuai dengan hukum yang berlaku.

Bandar Lampung, 21 April 2026

Theofani Hati Kusumawardani

NPM 2255061004

RIWAYAT HIDUP

Penulis lahir sebagai anak kedua dari tiga bersaudara dari pasangan Alexius Tri Hariyanto dan Esti Budi Handayani pada 13 Oktober 2004 di Kota Bandar Lampung. Penulis menyelesaikan pendidikan dasar di SDK BPK Penabur Metro pada tahun 2017 dan pendidikan menengah pertama di SMP Negeri 1 Metro pada tahun 2020. Penulis menempuh pendidikan menengah atas sebagai siswa percepatan di SMA Negeri 1 Metro dan selesai pada tahun 2022. Pada tahun 2022, penulis secara resmi terdaftar sebagai mahasiswa Program Studi Teknik Informatika Universitas Lampung. Selama menjadi mahasiswa, penulis aktif dalam kegiatan baik akademis maupun non-akademis. Penulis terlibat sebagai Asisten Laboratorium Teknik Digital dan aktif mengajar mulai tahun 2023 hingga 2026. Penulis terlibat aktif dalam kegiatan persekutuan dan menjadi pengurus Forum Komunikasi Mahasiswa Kristiani Fakultas Teknik (FKMK-FT) Generasi ke-24. Pengembangan diri tidak hanya penulis dapatkan melalui lingkup kampus, pada tahun 2024, penulis ikut dalam program Studi Independen Bersertifikat Kampus Merdeka di mitra Bangkit *by Google, GoTo, and Traveloka* dengan fokus bidang *Mobile Development*. Penulis juga berkesempatan mengerjakan *capstone project* di Bakrie Center Foundation, Jakarta Selatan, dalam tim *Mobile Developer LEAD Indonesia* dan meraih *Best Team Company* serta *Best 22 Company Track Capstone Project*. Pada tahun 2025, penulis berkesempatan melakukan praktik kerja di Dinas Kehutanan Provinsi Lampung sebagai *front end website developer*. Selain itu, penulis juga menerapkan ilmu yang didapat selama perkuliahan pada lingkup masyarakat dengan ikut serta dalam pelaksanaan kegiatan Kuliah Kerja Nyata (KKN) di Desa Purwodadi, Kecamatan Way Sulan, Kabupaten Lampung Selatan.

MOTTO

”If you don't fight for what you want, then don't cry for what you lost.”

Sri Krishna – Bhagavad Gita

”Aku tidak akan menangis karena sesuatu telah berakhir, tapi aku akan tersenyum karena sesuatu itu pernah terjadi”

Tere Liye – Tentang Kamu

”Waktunya makan, makan; waktunya tidur, tidur; waktunya kerja, kerja.”

E – rekan Katalis

“Tetaplah berdoa.”

1 Tesalonika 5:17

PERSEMBAHAN

Segala Puji Syukur kepada Tuhan Yang Maha Esa; Bapa, Putra, dan Roh Kudus, karena kasih dan setia-Nya yang senantiasa menyertai penulis dalam penelitian ini hingga terselesaikan.

Kupersembahkan Skripsi ini Kepada

Papa, Mama, Kak Visi, dan Jemy, yang tak henti-hentinya memberi dukungan doa, semangat, dan menuntun penulis hingga sampai sejauh ini. Kiranya kasih setia-Nya senantiasa menyertai setiap langkah dan damai-Nya melingkupi setiap hari.

Diriku di masa lalu dan masa depan, yang tak menyerah dan tetap takkan pernah menyerah walau ratusan nanar, ribuan rebah memamah.

SANWACANA

Puji syukur kepada Tuhan Yang Maha Esa, karena hanya dengan kasih, karunia, serta anugerah-Nya, penulis dapat menyusun penelitian berjudul "Penerapan *Test-Driven Development* pada Aplikasi Mobile Android Sistem Informasi Petani Hutan (Studi Kasus Dinas Kehutanan Provinsi Lampung)".

Penyusunan skripsi atau tugas akhir ini merupakan persyaratan yang harus dipenuhi dalam kurikulum Program Studi Teknik Informatika. Sehingga menjadi suatu sukacita serta kebahagiaan dapat melaksanakan penelitian ini dengan banyaknya ilmu, bimbingan, arahan, serta saran dari berbagai pihak. Untuk itu, penulis mengucapkan terima kasih kepada:

1. Ibu Helinawati S.T., M.T., selaku Ketua Jurusan Teknik Elektro, Universitas Lampung.
2. Ibu Yessi Mulyani S.T., M.T., selaku Ketua Program Studi Teknik Informatika, Universitas Lampung.
3. Bapak Wahyu Eko S, S.T., M.Sc., selaku Dosen Pembimbing Utama yang memberikan arahan serta masukan dalam penulisan tugas akhir.
4. Bapak Sony Ferbangkara, S.T., M.T, selaku Dosen Pembimbing Pendamping yang memberikan arahan serta masukan dalam penulisan tugas akhir.
5. Bapak Mona Arif Muda Batubara, S.T, M.T, selaku Dosen Penguji pada penelitian ini.
6. Kak Aldo selaku *project manager*, Mba Listi, Bapak Kepala Dinas, dan bapak ibu Tahura atas ilmu, kebersamaan, serta bimbingannya selama pelaksanaan penelitian di Dinas Kehutanan Provinsi Lampung.
7. Kak Visi, papa, mama, serta Jemy yang selalu memberikan dukungan, semangat, dan menghadirkan senyum penulis selama pelaksanaan dan penulisan penelitian tugas akhir.

8. Nathan, yang senantiasa menjadi tempat bersandar, berkeluh, berkesah, bercanda, hingga berangan bersama.
9. Semua pihak yang tidak dapat disebutkan namanya. Ucapan terima kasih penulis sampaikan atas doa dan dukungannya kepada penulis.

Penulis menyadari bahwa dalam penulisan penelitian ini masih terdapat kekurangan. Untuk itu, penulis terbuka atas kritik, saran, dan masukan yang membangun dari semua pihak untuk meningkatkan kualitas laporan ini. Kiranya penelitian tugas akhir ini bermanfaat bagi kita semua.

Bandar Lampung, 31 Maret 2026
Penulis,

Theofani Hati Kusumawardani
NPM 2255061004

DAFTAR ISI

Halaman
DAFTAR ISI iv
DAFTAR TABEL x
DAFTAR GAMBAR xii
I. PENDAHULUAN 16
1.1. Latar Belakang 16
1.2. Rumusan Masalah 18
1.3. Tujuan Penelitian 18
1.4. Manfaat Penelitian 19
1.5. Batasan Masalah 19
1.6. Sistematika Penulisan 20
II. TINJAUAN PUSTAKA 21
2.1. Teori Terkait 21
2.1.1. Nilai Transaksi Ekonomi 21
2.1.2. Mobile Application 22
2.1.3. Clean Architecture 24
2.1.4. Personal Extreme Programming (PXP) 25
2.1.5. Test Driven Development (TDD) 27
2.1.6. Moscow Prioritization 29
2.1.7. Offline-first application 30
2.1.8. Unified Modeling Language 31
2.1.8.1. Use Case Diagram 32
2.1.8.2. Activity Diagram 33
2.1.9. Unit testing 34
2.1.10. Cyclomatic Complexity 36

2.1.11. Cognitive Complexity 37
2.1.12. Black box testing 40
2.1.13. User Acceptance Test (UAT) 40
2.2. Penelitian Terkait 41
III. METODE PENELITIAN 45
3.1. Waktu dan Tempat 45
3.1.1. Waktu Penelitian 45
3.1.2. Tempat Penelitian 45
3.2. Alat dan Bahan Penelitian 46
3.2.1. Alat Penelitian 46
3.2.2. Bahan Penelitian 46
3.3. Tahapan Penelitian 48
3.3.1. Studi Literatur 50
3.3.2. Requirement 50
3.3.3. Planning 50
3.3.4. Iterasi 50
3.3.4.1. Iteration Initialization 50
3.3.4.2. Design 51
3.3.4.3. Implementation 51
3.3.4.4. System testing 51
3.3.4.5. Retrospective 52
3.3.5. Pengujian pengguna 52
3.3.6. Penyerahan Proyek 53
IV. HASIL DAN PEMBAHASAN 54
4.1. Hasil 54
4.1.1. Studi Literatur 54
4.1.2. Requirement 54

4.1.2.1. Kebutuhan Fungsional 56
4.1.2.2. Kebutuhan Non Fungsional 58
4.1.2.3. User Stories 59
4.1.3. Planning 63
4.1.3.1. Prioritas User Stories 63
4.1.3.2. Use Case Diagram 65
4.1.3.3. Architecture Diagram 68
4.1.4. Iterasi 1 69
4.1.4.1. Iteration Initialization 69
4.1.4.2. Design 69
a. Entity Relationship Diagram 70
b. Activity Diagram 71
c. High Fidelity Design 76
4.1.4.3. Implementation 79
a. Struktur Proyek 81
b. Unit Testing 82
c. Code 84
d. Refactor 86
e. Hasil Antarmuka 88
4.1.4.4. System Testing 91
4.1.4.5. Retrospective 93
4.1.5. Iterasi 2 94
4.1.5.1. Iteration Initialization 94
4.1.5.2. Design 95
a. Activity Diagram 95
b. High Fidelity Design 101

4.1.5.3. Implementation 103
a. Unit Testing 103
b. Code 104
c. Refactor 106
d. Hasil Antarmuka 107
4.1.5.4. System Testing 109
4.1.5.5. Retrospective 109
4.1.6. Iterasi 3 111
4.1.6.1. Iteration Initialization 111
4.1.6.2. Design 111
a. Activity Diagram 111
b. High Fidelity Design 117
4.1.6.3. Implementation 119
a. Unit Testing 119
b. Code 120
c. Refactor 121
d. Hasil Antarmuka 123
4.1.6.4. System Testing 124
4.1.6.5. Retrospective 124
4.1.7. Iterasi 4 125
4.1.7.1. Iteration Initialization 125
4.1.7.2. Design 126
a. Activity Diagram 126
b. High Fidelity Design 131
4.1.7.3. Implementation 132
a. Unit Testing 132

b. Code 133
c. Refactor 134
d. Hasil Antarmuka 136
4.1.7.4. System Testing 137
4.1.7.5. Retrospective 137
4.1.8. Iterasi 5 138
4.1.8.1. Iteration Initialization 138
4.1.8.2. Design 139
a. Activity Diagram 139
b. High Fidelity Design 144
4.1.8.3. Implementation 146
a. Unit Testing 146
b. Code 148
c. Refactor 149
d. Hasil Antarmuka 150
4.1.8.4. System Testing 152
4.1.8.5. Retrospective 152
4.1.9. Iterasi 6 153
4.1.9.1. Iteration Initialization 153
4.1.9.2. Design 154
a. Activity Diagram 154
b. High Fidelity Design 159
4.1.9.3. Implementation 161
a. Test 161
b. Code 163
c. Refactor 164

d. Hasil Antarmuka 166
4.1.9.4. System Testing 167
4.1.9.5. Retrospective 169
4.1.10. Pengujian Pengguna 172
4.1.11. Penyerahan Proyek 175
4.2. Pembahasan 176
4.2.1. Fungsionalitas Offline-First 176
4.2.2. Penerapan Test Driven Development (TDD) 177
4.2.3. Pemenuhan kebutuhan aplikasi 179
V. SIMPULAN DAN SARAN 180
5.1. Simpulan 180
5.2. Saran 181
DAFTAR PUSTAKA 183

DAFTAR TABEL

Tabel Halaman

Tabel 1. Notasi Use Case Diagram 32
Tabel 2. Notasi Activity Diagram..... 33
Tabel 3. Waktu Pelaksanaan Penelitian..... 45
Tabel 4. Perangkat Keras..... 46
Tabel 5. Perangkat Lunak..... 46
Tabel 6. Kebutuhan Aplikasi..... 55
Tabel 7. Kebutuhan Fungsional Sistem..... 56
Tabel 8. Kebutuhan non-Fungsional Sistem 58
Tabel 9. User Stories 59
Tabel 10. Prioritas User Stories..... 63
Tabel 11. Task iterasi 1 69
Tabel 12. Pengujian unit iterasi 1 83
Tabel 13. Test case iterasi 1..... 92
Tabel 14. Evaluasi estimasi dan realisasi iterasi 1 93
Tabel 15. Logs iterasi 1 93
Tabel 16. Task Iterasi 2 94
Tabel 17. Pengujian Unit Test Iterasi 2 104
Tabel 18. Evaluasi estimasi dan realisasi iterasi 2 110
Tabel 19. Logs iterasi 2 110
Tabel 20. Task Iterasi 3 111
Tabel 21. Pengujian Unit Test Iterasi 3 120
Tabel 22. Evaluasi estimasi dan realisasi iterasi 3 124
Tabel 23. Logs iterasi 3 125
Tabel 24. Task Iterasi 4 125
Tabel 25. Pengujian Unit Test Iterasi 4 133
Tabel 26. Evaluasi estimasi dan realisasi iterasi 4 137

Tabel 27. Logs iterasi 4 138
Tabel 28. Task Iterasi 5 138
Tabel 29. Pengujian Unit Test Iterasi 5 147
Tabel 30. Evaluasi estimasi dan realisasi iterasi 5 152
Tabel 31. Logs iterasi 5 153
Tabel 32. Task Iterasi 6 153
Tabel 33. Pengujian Unit Test Iterasi 6 162
Tabel 34. Evaluasi black box seluruh fitur 167
Tabel 35. Evaluasi code coverage 168
Tabel 36. Evaluasi Cyclomatic dan Cognitive Complexity 169
Tabel 37. Evaluasi estimasi dan realisasi iterasi 6 170
Tabel 38. Evaluasi estimasi dan realisasi total 170
Tabel 39. Evaluasi kesesuaian pengembangan dan kebutuhan 172
Tabel 40. Hasil pengujian pengguna 173

DAFTAR GAMBAR

Gambar Halaman

Gambar 1. Layer dalam clean architecture [8] 24
Gambar 2. Fase PXP [20] 26
Gambar 3. Siklus TDD 28
Gambar 4. Moscow prioritization 29
Gambar 5. Struktur pustaka Room [6] 31
Gambar 6. Tahapan Penelitian 48
Gambar 7. Use Case Diagram Aktor Petani 65
Gambar 8. Use Case Diagram Aktor Penyuluh dan penanggung jawab 66
Gambar 9. Use Case Diagram Aktor Penyuluh dan penanggung jawab (lanjutan) 67
Gambar 10. Sitanihut Architecture Diagram 68
Gambar 11. Entity Relationship Diagram 70
Gambar 12. Activity diagram masuk akun 71
Gambar 13. Activity diagram mengubah kata sandi melalui Lupa Password 72
Gambar 14 .Activity diagram keluar dari akun 73
Gambar 15. Activity Diagram Melihat Informasi 74
Gambar 16. Activity diagram melihat profil 75
Gambar 17. Antarmuka landing page (a)halaman 1 (b)halaman 2 (c)halaman 3 . 76
Gambar 18. Antarmuka autentikasi (a)login (b)forgot password 76
Gambar 19. Antarmuka beranda (a)penyuluh (b)penanggung jawab (c)petani 77
Gambar 20. Antarmuka profil pengguna (a)petani (b)penyuluh (c)kepala KPH .. 78
Gambar 21. Antarmuka (a)informasi (b)hubungi kami 78
Gambar 22. Antarmuka informasi (a)tentang instansi (b)tentang aplikasi 79
Gambar 23. Struktur Proyek Sitanihut 81
Gambar 24. Test LoginViewModel layer presentation FAIL 82
Gambar 25. Kode LoginViewModel layer presentation 84

Gambar 26. Test LoginViewModel layer presentation PASS 85
Gambar 27. View model login sebelum refactoring 86
Gambar 28. View model login setelah refactoring..... 86
Gambar 29. Pengujian setelah refactoring 87
Gambar 30. Implementasi antarmuka Landing Page 88
Gambar 31. Implementasi antarmuka (a)login (b)forgot password 88
Gambar 32. Implementasi antarmuka homepage (a)petani (b)penyuluh (c)logout 89
Gambar 33. Implementasi antarmuka profil (a)petani (b)penyuluh..... 89
Gambar 34. Implementasi antarmuka (a)informasi (b)tentang aplikasi..... 90
Gambar 35. Implementasi antarmuka (a)kontak (b)tentang instansi 90
Gambar 36. Activity Diagram Melihat Daftar Laporan 96
Gambar 37. Membuat laporan baru 97
Gambar 38.Activity diagram melihat detail laporan..... 98
Gambar 39. Activity Diagram Mengedit dan Menghapus Laporan 99
Gambar 40. Activity Diagram Melihat Daftar Komoditas..... 100
Gambar 41. Antarmuka (a)data komoditas (b)daftar laporan (c) detail laporan . 101
Gambar 42. Antarmuka (a)tambah laporan (b)edit laporan 102
Gambar 43. Test ReportDetailViewModel layer presentation FAIL..... 103
Gambar 44. Kode ReportDetailViewModel..... 104
Gambar 45. Test ReportDetailViewModel PASS..... 105
Gambar 46. Kode ReportDetailViewModel Sebelum Refaktor..... 106
Gambar 47. Kode ReportDetailViewModel Setelah Refaktor..... 106
Gambar 48. Pengujian setelah refaktor 107
Gambar 49. Implementasi (a)list komoditas (b)list laporan (c)filter laporan 107
Gambar 50. Implementasi laporan (a)detail (b)tambah (c)edit..... 108
Gambar 51. Implementasi pesan konfirmasi (a)pengajuan (b)hapus..... 108
Gambar 52. Activity Diagram Melihat daftar laporan petani 112
Gambar 53. Activity Diagram Memeriksa laporan (penyuluh) 113
Gambar 54. Activity Diagram Memeriksa laporan (penanggung jawab) 114
Gambar 55. Activity Diagram Melihat daftar penyuluh 115
Gambar 56. Activity Diagram Melihat detail penyuluh.....116

Gambar 57. Antarmuka (a)daftar laporan (b)daftar penyuluh (c) detail penyuluh 117
Gambar 58. Antarmuka periksa laporan (a)penyuluh (b)penanggung jawab 118
Gambar 59. Test ReportListViewModel layer presentation FAIL 119
Gambar 60. Kode ReportListViewModel 120
Gambar 61. Test ReportListViewModel PASS 121
Gambar 62. Kode ReportListViewModel Sebelum Refaktor 121
Gambar 63. Kode ReportListViewModel Setelah Refaktor 122
Gambar 64. Pengujian setelah refaktor 122
Gambar 65. Implementasi periksa laporan (a)daftar (b)filter (c) detail 123
Gambar 66. Implementasi periksa laporan (a)penyuluh (b)penanggung jawab . 123
Gambar 67. Activity Diagram Melihat daftar KTH 126
Gambar 68. Activity Diagram Menambah data KTH baru 127
Gambar 69. Activity Diagram Melihat Detail KTH..... 128
Gambar 70. Activity Diagram Mengedit informasi KTH 129
Gambar 71. Activity Diagram Menghapus KTH 130
Gambar 72. Antarmuka KTH (a)list (b)tambah data 131
Gambar 73. Antarmuka KTH (a)edit data (b)detail data..... 131
Gambar 74. Test KthFormViewModel layer presentation FAIL..... 132
Gambar 75. Test KthFormViewModel PASS..... 133
Gambar 76. Kode KthFormViewModel Sebelum Refaktor..... 134
Gambar 77. Kode KthFormViewModel Setelah Refaktor..... 134
Gambar 78. Pengujian setelah refaktor 135
Gambar 79. Implementasi (a)daftar KTH (b)aksi (c)detail KTH 136
Gambar 80. Implementasi KTH (a)tambah data (b)edit (c)pesan konfirmasi 136
Gambar 81. Activity Diagram Melihat daftar petani 139
Gambar 82. Activity Diagram Menambah data petani baru 140
Gambar 83. Activity Diagram Melihat detail petani..... 141
Gambar 84. Activity Diagram Mengedit data petani 142
Gambar 85. Activity Diagram Menghapus data petani..... 143
Gambar 86. Antarmuka data petani (a)list (b)detail..... 144
Gambar 87. Antarmuka data petani (a)tambah data (b)edit data 145

Gambar 88. Test ValidatePetaniInputUseCase layer presentation FAIL 146
Gambar 89. Kode ValidatePetaniInputUseCase 148
Gambar 90. Test ValidatePetaniInputUseCase PASS 148
Gambar 91. Kode ValidatePetaniInputUseCase Sebelum Refaktor 149
Gambar 92. Kode ValidatePetaniInputUseCase Setelah Refaktor 149
Gambar 93. Pengujian setelah refaktor 150
Gambar 94. Implementasi data petani (a)list (b)aksi (c)detail 151
Gambar 95. Implementasi data petani (a)ekspor (b)tambah (c)edit 151
Gambar 96. Pesan konfirmasi (a)simpan (b)perbarui (c)hapus 152
Gambar 97. Activity Diagram Melihat daftar akun pengguna 154
Gambar 98. Activity Diagram Membuat akun petani baru 155
Gambar 99. Activity Diagram Melihat detail pengguna 156
Gambar 100. Activity Diagram Mengedit akun pengguna 157
Gambar 101. Activity Diagram Menghapus pengguna 158
Gambar 102. Antarmuka kelola pengguna (a)list (b)detail 159
Gambar 103. Antarmuka kelola pengguna (a)tambah (b)edit 160
Gambar 104. Test GetUserListUseCase layer domain FAIL 161
Gambar 105. Kode GetUserListUseCase 163
Gambar 106. Test GetUserListUseCase PASS 163
Gambar 107. Kode GetUserListUseCase Sebelum Refaktor 164
Gambar 108. Kode GetUserListUseCase Setelah Refaktor 164
Gambar 109. Pengujian setelah refaktor 165
Gambar 110. Implementasi kelola pengguna (a)list (b)opsi aksi (c)detail 166
Gambar 111. Implementasi kelola pengguna (a)tambah (b)edit (c)konfirmasi .. 166
Gambar 112. Presentasi dan penyerahan proyek pada kepala dinas 175

I. PENDAHULUAN

1.1. Latar Belakang

Nilai Transaksi Ekonomi (NTE) merupakan indikator yang merefleksikan vitalitas ekonomi dari sektor kehutanan dan pertanian. NTE tidak hanya mengukur volume produksi, namun juga menggambarkan nilai yang dihasilkan dari pemanfaatan sumber daya hutan oleh masyarakat, khususnya para petani hutan. Bagi instansi, dalam hal ini Dinas Kehutanan, data NTE yang akurat menjadi landasan untuk pengambilan kebijakan, alokasi sumber daya, serta evaluasi efektivitas program pemberdayaan masyarakat.

Guna memfasilitasi pengumpulan data pelaporan petani hutan, sebuah aplikasi berbasis *website* bernama Sistem Informasi Petani Hutan (Sitanihut) telah dikembangkan. *Website* Sitanihut dibangun guna membantu proses pencatatan dan pelaporan hasil tanam, panen, serta NTE yang didapatkan. Sistem pencatatan NTE yang dimaksud tidak serta merta mendukung aksi pemanfaatan hutan secara berlebihan, tetapi justru menjadi metode pengawasan dalam pemanfaatan hasil hutan. Pencatatan dan pelaporan yang baik menjadi salah satu mekanisme untuk memantau pemanfaatan hutan agar tidak keluar batas. Hal ini sejalan dengan visi misi Kementerian Lingkungan Hidup dan Kehutanan (KLHK) di tingkat nasional bahwa akan menyeimbangkan ekologi, ekonomi, sosial dan mendukung pembangunan ekonomi hijau [1]. Adapun pencatatan NTE yang dilakukan digunakan untuk transaksi HHBK (Hasil Hutan Bukan Kayu) [2] dan dilakukan oleh Kelompok Tani Hutan (KTH) yang telah teregistrasi dan disetujui oleh KLHK sebagai perhutanan sosial.

Meski *website* Sitanihut telah dikembangkan, aksesbilitas melalui perangkat *mobile* menjadi kebutuhan, karena pengguna *smartphone* jauh lebih tinggi dibandingkan laptop. Berdasarkan Berita Resmi Statistik dari Badan Pusat Statistik Provinsi Lampung April tahun 2024, persentase rumah tangga di Lampung yang memiliki/menguasai komputer sebesar 10,63% [3]. Sementara Badan Pusat Statistik (BPS) Indonesia mencatat persentase rumah tangga yang memiliki/menguasai telepon seluler di Lampung ialah 96,66% di perkotaan dan 94,85% di pedesaan pada tahun 2024 [4]. Kepemilikan telepon seluler jauh melampaui kepemilikan komputer di Indonesia.

Tantangan utama yang dihadapi pengguna di lapangan ialah konektivitas jaringan yang tidak merata. Berdasarkan data Podes BPS Provinsi Lampung tahun 2024, sebanyak 365 kantor desa/kelurahan di Lampung belum memiliki fasilitas internet. Selain itu, di 75 desa fasilitasnya tidak berfungsi dan di 48 desa lainnya jarang berfungsi [5]. Hal ini membuat fungsionalitas *offline,* yang mampu beroperasi di daerah-daerah dengan koneksi rendah bahkan tanpa koneksi, diperlukan.

Aplikasi *mobile* menggunakan penyimpanan perangkat dan memiliki *local database* untuk manajemen *offline*, dengan menggunakan pustaka Room berbasis SQLite [6]. Dengan *local database* tersebut, aplikasi dapat menyimpan data dalam database dan mengunggahnya secara otomatis ke server ketika koneksi tersedia.

Namun demikian, fungsionalitas dan manajemen data *offline* serta logika sinkronisasi otomatis memiliki tingkat kompleksitas yang tinggi. Sebuah sistem yang memiliki banyak percabangan, alur, maupun *nesting* membuat kode semakin kompleks dan lebih sulit untuk diuji dan dipelihara [7]. Hal ini berdampak pada *maintainability* sistem dan meningkatkan risiko *bug* saat penambahan fitur. Pengujian dalam TDD bersifat terisolasi per unitnya, sehingga menjadi pengaman ketika dilakukan penambahan fitur baru. Apabila terdapat penambahan fitur di masa depan, rangkaian tes tersebut dapat dijalankan kembali untuk memastikan bahwa kode baru tidak merusak fungsionalitas yang sudah ada. Adapun kode yang mengimplementasikan TDD memiliki *code coverage* yang lebih tinggi [7].

Oleh karena itu, diperlukan pengembangan aplikasi versi *mobile* dari website Sitanihut yang menyediakan fungsionalitas *offline-first*. Sebagaimana tujuannya guna pemakaian lapangan, versi mobile dari Sitanihut ini mengakomodasi peran petani, penyuluh, dan penanggung jawab, sementara pengaksesan auditor dan sekretariat dilakukan melalui *website* karena kebutuhan penampilan data secara lengkap. Dengan adanya fungsionalitas *offline* dan sinkronisasi data yang membuatnya kompleks, aplikasi dikembangkan dengan menerapkan metode *Test-Driven Development* (TDD) guna memastikan unit logika teruji secara terisolasi. Pun aplikasi dikembangkan dengan arsitektur *Clean Architecture* untuk meningkatkan *maintainability*, sehingga dengan pemisahan *layer* yang ada [8], perubahan maupun penambahan fitur di masa mendatang dapat dilakukan tanpa menyebabkan konflik antar komponen. Namun penelitian tidak menguji arsitektur tersebut secara khusus.

1.2.Rumusan Masalah

Adapun rumusan masalah dalam penelitian ini adalah sebagai berikut.

1. Bagaimana membangun aplikasi *mobile* android yang memiliki fungsionalitas *offline-first* dengan menerapkan *local database storage*?
2. Bagaimana *Test Driven Development* diterapkan pada pengembangan aplikasi *mobile* dengan fungsionalitas *offline* guna menjaga *maintainability* sistem?
3. Apakah aplikasi *mobile* yang dikembangkan telah memenuhi kebutuhan dan dapat digunakan dengan baik oleh pengguna terkait pelaporan data Nilai Transaksi Ekonomi (NTE) petani hutan?

1.3.Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah sebagai berikut.

1. Membangun aplikasi *mobile* android yang memiliki fungsionalitas *offline-first* dengan *local database storage*.
2. Menerapkan *Test Driven Development* pada pengembangan aplikasi *mobile offline-first* untuk menjaga *maintainability* sistem.

3. Menguji fungsionalitas aplikasi *mobile* Sitanihut agar sesuai kebutuhan pengguna terkait pelaporan data Nilai Transaksi Ekonomi (NTE) petani hutan dalam naungan Dinas Kehutanan Provinsi Lampung.

1.4. Manfaat Penelitian

Adapun manfaat dari penelitian ini adalah sebagai berikut.

1. Bagi petani hutan dan penyuluh kehutanan

   Penelitian ini berkontribusi pada pengembangan aplikasi Sitanihut yang membantu proses pencatatan dan pelaporan kegiatan pertanian hutan dan Nilai Transaksi Ekonomi (NTE) yang dihasilkan oleh petani hutan. Bagi penyuluh, penelitian membantu proses perekapan data dari petani dan pelaporan kepada instansi.
2. Bagi instansi Dinas Kehutanan Provinsi Lampung

   Mempermudah pencatatan digital dan memberikan data rekapitulasi mengenai NTE petani hutan, yang dapat digunakan sebagai dasar pengambilan keputusan atau perumusan kebijakan.
3. Bagi akademisi maupun peneliti selanjutnya

   Hasil penelitian ini dapat menjadi dasar bagi pengembangan lebih lanjut di masa depan, terutama terkait *Test Driven Development* (TDD) dan pengembangan fungsional *offline-first* pada aplikasi mobile android. Termasuk juga penelitian terkait *maintainability* dari penerapan TDD baik secara konseptual maupun praktik.
4. Bagi masyarakat luas

   Penelitian ini dapat menjadi sumber referensi pembelajaran bagi masyarakat luas.

1.5.Batasan Masalah

Adapun batasan masalah dalam penelitian ini adalah sebagai berikut.

1. Penelitian fokus pada pengembangan *front end mobile* berdasarkan aplikasi berbasis *website* Sitanihut yang telah ada.

2. Penelitian ini mengembangkan aplikasi *mobile* dengan memanfaatkan API yang telah tersedia tanpa ada penambahan fitur diluar API tersebut.
3. Penelitian fokus hanya pada perangkat android.
4. Perancangan desain UI/UX diserahkan pada pengembang menyesuaikan dengan fungsionalitas yang dibutuhkan.

1.6. Sistematika Penulisan

Adapun struktur penulisan yang diterapkan pada penelitian ini sebagai berikut.

BAB I: PENDAHULUAN

Bab ini berisi latar belakang masalah, identifikasi masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II: TINJAUAN PUSTAKA

Bab ini berisi tentang penelitian-penelitian sebelumnya dan teori-teori yang mendukung sebagai referensi dalam penelitian ini.

BAB III: METODOLOGI PENELITIAN

Bab ini berisi tentang pendekatan dan metodologi yang digunakan dalam penelitian, mencangkup waktu dan tempat penelitian, alat dan bahan yang digunakan dalam penelitian, prosedur, serta analisis kebutuhan perangkat lunak.

BAB IV: HASIL DAN PEMBAHASAN

Bab ini berisi pembahasan mengenai penelitian yang dilakukan.

BAB V: SIMPULAN DAN SARAN

Bab ini memuat kesimpulan berdasarkan hasil penelitian dan saran yang diharapkan untuk pengembangan penelitian selanjutnya.

DAFTAR PUSTAKA

II. TINJAUAN PUSTAKA

2.1. Teori Terkait

2.1.1. Nilai Transaksi Ekonomi

Berdasarkan Rencana Kerja BP2SDM 2025, Nilai Transaksi Ekonomi (NTE) mengacu pada nilai rupiah yang dihasilkan dari aktivitas usaha kelompok tani hutan. Nilai tersebut dihitung berdasarkan perolehan omzet usaha, yakni perkalian jumlah produk terjual dengan harga per satuannya. NTE dihitung sebagai data perkiraan dan ukuran kesejahteraan masyarakat sekitar hutan serta sebagai pertimbangan dalam penyusunan kebijakan penyuluhan dan pendampingan [9]. Adapun NTE dihitung secara rutin dalam setiap bulannya dari tiap Kelompok Tani Hutan (KTH). Adapun perhitungan NTE dipaparkan sebagai berikut.

$$NTE = Jumlah\ produk\ terjual\ x\ harga\ jual/buah \quad (1)$$

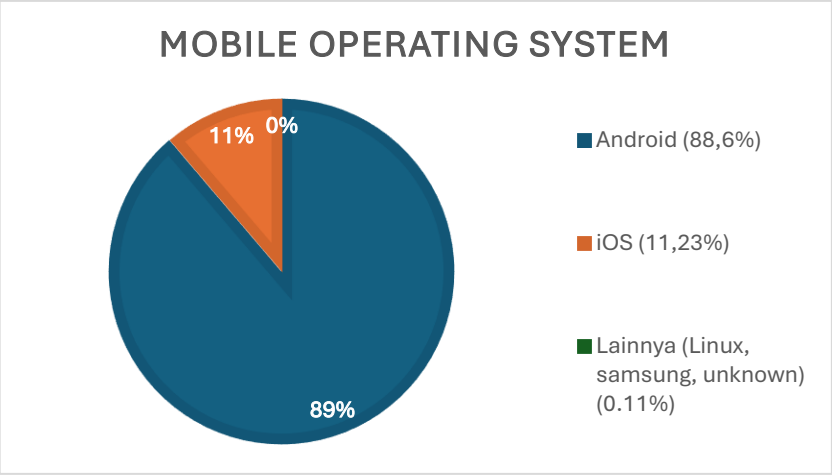
KTH mengacu pada kelompok tani hutan yang terdiri atas kumpulan petani hutan dalam kawasan yang sama. Setiap KTH dibina oleh Kesatuan Pengelolaan Hutan (KPH). Kesatuan Pengelolaan Hutan (KPH) ialah wilayah pengelolaan hutan yang dikelola secara lestari sebagai bagian dari sistem pengelolaan hutan nasional [10]. Saat ini, terdapat total 17 KPH di bawah binaan Dinas Kehutanan Provinsi Lampung [2].

2.1.2. Mobile Application

Dengan popularitas perangkat seluler dan basis pengguna yang besar, pengembangan aplikasi *mobile* menjadi *focal point* dalam rekayasa perangkat lunak [11]. Salah satunya ialah pengembangan *native*, yakni proses pembuatan aplikasi secara spesifik untuk satu platform menggunakan perangkat dan bahasa pemrograman yang dirancang khusus untuk platform tersebut [12]. Karakteristik utama aplikasi *native* adalah pemanfaatan *Software Development Kit* (SDK) resmi yang disediakan oleh vendor platform. Hal ini membuat aplikasi dapat menggunakan fitur terbaru dan mengikuti pembaruan API dari sistem operasi tersebut [12].

Native memiliki kelebihan dalam segi akses *hardware*, performa, dan kemampuan *offline* aplikasi. Aplikasi *native* memiliki kontrol penuh atas API platform [12] dan akses pada fungsionalitas perangkat. Studi menunjukkan bahwa penggunaan *framework cross-platform* cenderung menimbulkan *overhead* tambahan dan menurunkan performa aplikasi dibandingkan *native* [12].

Selain itu, aplikasi *native* memiliki sistem penyimpanan data *offline* yang aman karena data yang tersimpan dalam direktori internal sepenuhnya terikat pada aplikasi [13]. Adapun *Room persistence library* memungkinkan penyimpanan data persisten yang datanya terisolasi [6]. Namun, *native* memiliki kekurangan pada segi biaya pengembangan dan kebutuhan spesialisasi pengetahuan dalam pengembangannya [12]. Pengembang harus memahami secara khusus pengembangan aplikasi *native* untuk tiap *platform*.



Gambar 1. Jenis sistem operasi mobile[14]

Berdasarkan data dari StatCounter seperti ditunjukkan pada gambar 1, di Indonesia, Android merupakan sistem operasi *smartphone* yang memiliki pangsa pasar terbesar. Pangsa pasar Android mencapai 88.66% pada Juli 2025 [14]. Pada Google I/O 2019, Google menyatakan Kotlin sebagai rekomendasi bahasa pemrograman untuk pengembangan Android. Hal ini didasari karena Kotlin lebih ringkas, memiliki *null-safety* bawaan untuk mencegah *common programming mistakes* dalam pengembangan. Selain itu, kotlin juga memiliki *coroutines* untuk menyederhanakan penulisan kode asinkron untuk *background task* [15]. Berdasarkan studi, migrasi dari Java ke Kotlin memiliki pengaruh yang signifikan secara statistik terhadap penggunaan CPU dan memori [16].

Untuk pembangunan antarmuka pengguna (UI), pendekatan modern dalam ekosistem Android beralih ke Jetpack Compose. Compose ialah *toolkit* UI modern dari Google untuk membangun antarmuka aplikasi Android *native* secara deklaratif. Keunggulannya ialah *less code*, mengurangi jumlah kode yang diperlukan untuk membangun dan memelihara UI dibandingkan sistem View berbasis XML. Serta intuitif, yakni UI secara otomatis diperbarui dengan *Live Preview* tanpa perlu menjalankan ulang aplikasi [17].

2.1.3. Clean Architecture

Clean Architecture adalah pendekatan yang diusulkan oleh Robert C. Martin. Menekankan prinsip *Separation of Concerns* (SoC) dengan membagi sistem dalam lapisan/*layer* independen [8]. Bertujuan melindungi *layer* dalam dari *layer-layer* di atasnya. *Dependency Rule* bahwa semua ketergantungan kode sumber mengarah ke dalam, yaitu dari lapisan luar (detail implementasi) menuju lapisan dalam (*business rules*). Sehingga perubahan pada komponen eksternal seperti UI atau *database* tidak akan memengaruhi inti logika bisnis.

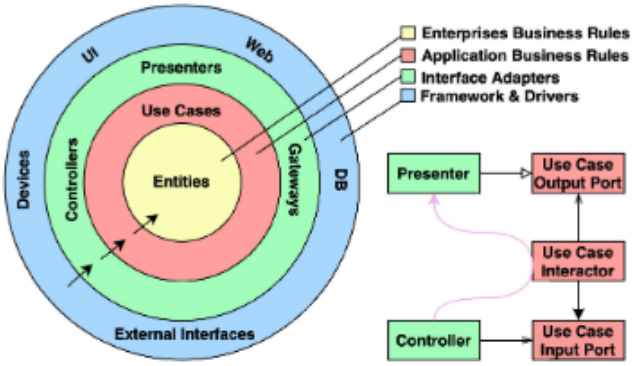


FIGURE 4. **Clean architecture [37].**

Gambar 2. Layer dalam clean architecture [8]

Gambar 2 memaparkan struktur *Clean Architecture* yang memiliki lapisan-lapisan, mencangkup *entities, use case, interface adapter,* dan *frameworks & drivers*. Seperti dijelaskan sebagai berikut[8].

- *Entities*

Entity merupakan objek bisnis dari aplikasi. Berisi aturan bisnis *enterprise-level (Enterprise Business Rules).* Berisi aturan bisnis umum dan *high-level rules*. Memiliki kemungkinan paling kecil untuk berubah ketika layer eksternal berubah.

- *Use Case*

Berisi aturan bisnis spesifik aplikasi (*Application Business Rules*). Layer ini mengimplementasikan *use case* sistem. Mengatur *data flow* dari atau menuju *entity.*

- *Interface Adapter*

Sebagai konverter data. *Presenter* atau *Controller* di dalamnya mengubah data dari format yang sesuai untuk lapisan luar (contohnya *database*) menjadi format yang dapat digunakan oleh *Use Cases* dan *Entities*, berlaku sebaliknya.

- *Frameworks* dan *Drivers*

Lapisan terluar berisi detail implementasi dan elemen eksternal. Termasuk UI (*User Interface*), *database*, *web framework*, dan lainnya.

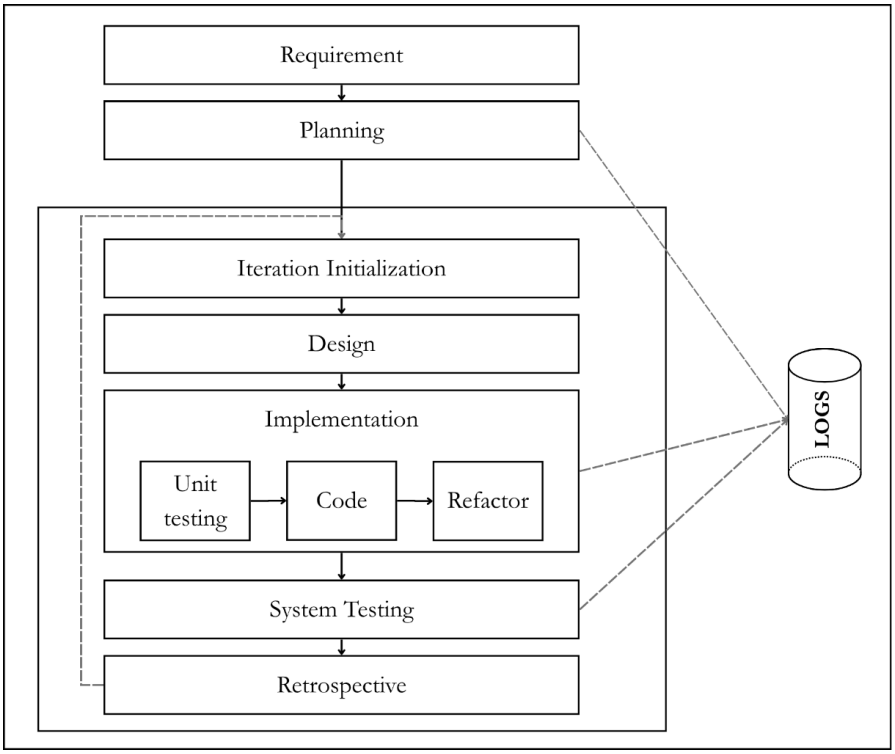
Clean Architecture memiliki kelebihan pada *maintainability*. Seiring perkembangannya, sistem memerlukan *maintenance cost* terutama pada *spelunking*, yakni penelusuran sistem yang telah ada guna menambahkan fitur baru, dan *risk*, yakni risiko terjadinya kerusakan sengaja maupun tidak sengaja dalam pengembangan suatu aplikasi [8]. *Clean Architecture* memisahkan sistem dalam *layer-layer* dan mengisolasinya sehingga mengurangi risiko kerusakan ketika pengembangan lebih lanjut.

Arsitektur yang dirancang dengan cermat sangat mengurangi biaya-biaya ini. Dengan memisahkan sistem menjadi beberapa komponen, dan mengisolasi komponen-komponen tersebut melalui antarmuka yang stabil, dimungkinkan untuk menerangi jalur bagi fitur-fitur di masa mendatang dan sangat mengurangi risiko kerusakan yang tidak disengaja.

2.1.4. Personal Extreme Programming (PXP)

PXP merupakan merupakan salah satu metode agile yang dikembangkan dari hasil kombinasi antara *Personal Software Process* (PSP) dan *Extreme Programming* (XP). *Personal Software Process* (PSP) ialah metode pengembangan perangkat lunak yang dibuat untuk pengembang individual. PSP menekankan pengukuran terhadap produk yang dihasilkan dan kualitasnya [18]. Namun, PSP memerlukan pemahaman mendalam tentang spesifikasi prosesnya dan menuntut komitmen tinggi. PXP memungkinkan *developer* mengembangkan sistem secara individu tanpa terikat pada suatu tim [19]. Metode ini mengadaptasi praktik-praktik XP,

termasuk *Test-Driven Development* (TDD) dalam fase pengembangannya, seperti dipaparkan sebagai berikut [20].



Gambar 3. Fase PXP [20]

Gambar 3 memaparkan bahwa pengembangan PXP bersifat iteratif, sehingga pengembang menjadi lebih fleksibel terhadap perubahan yang terjadi selama proyek berlangsung.

Proses PXP terdiri dari beberapa tahapan, yakni *Requirements, Planning, Iteration Initialization, Design, Implementation, Testing,* dan *Retrospective*. Adapun dijelaskan sebagai berikut.

- *Requirements*

  Fase identifikasi kebutuhan pengguna terkait sistem yang akan dikembangkan. Berdasarkan hasil pengumpulan kebutuhan tersebut kemudian dibuat *user stories*, yakni penjelasan ringkas fitur-fitur dengan sudut pandang pengguna.

- *Plannings*

  Pengembang mendefinisikan tugas/*task* berdasarkan *user stories* yang ada untuk kemudian diimplementasikan pada iterasi. Kemudian dibuat perkiraan waktu yang akan dihabiskan untuk setiap tugas.

- *Iteration Inizialitation*

  Menentukan tugas mana yang akan menjadi fokus untuk iterasi tersebut. Setiap iterasi memiliki tujuan utama yang berbeda, penentuan prioritas didasarkan pada prioritas *user story* pada tahap *planning*.
- *Design*

  Membuat model sistem berdasarkan kebutuhan yang didapatkan.
- *Implementation*

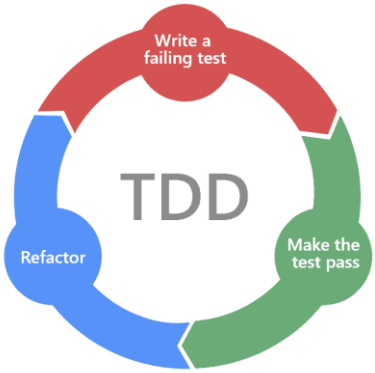
  Menerapkan *Test Driven Development* (TDD), terdapat 3 sub-fase, yakni *Unit testing*, *Code*, dan *Refactoring*. Sebelum membangun kode, pengembang terlebih dahulu membuat *script test* pada unit (*unit testing*), kemudian baru mengembangkan fungsionalitasnya hingga lulus uji (*code*), kemudian memperbaiki struktur atau menambahkan kode tanpa mengubah fungsionalitas yang telah terbentuk (*refactoring*).
- *System Testing*

  Pengujian yang dilakukan setelah pengembangan fitur iterasi tersebut selesai.
- *Retrospective*

  Dilakukan analisis terhadap data yang terkumpul selama fase-fase sebelumnya. Pengembang melakukan verifikasi apakah estimasi waktu pengerjaan tugas sesuai dengan waktu aktual yang dihabiskan untuk perbaikan di iterasi selanjutnya.

2.1.5. Test Driven Development (TDD)

Test Driven Development (TDD) ialah pendekatan pengembangan perangkat lunak dengan menulis pengujian terlebih dahulu sebelum penulisan fungsionalitasnya. Dicetuskan oleh Kent Beck pada 2003, TDD memiliki dua aturan, yakni menulis kode baru hanya jika tes gagal dan kemudian menghilangkan semua duplikasi kode[21]. Pun salah satu karakteristik dari tes dalam TDD adalah setiap tes terisolasi/*isolated* dari tes lainnya agar tidak ada ketergantungan antar pengujian.



Gambar 4. Siklus TDD
(gambar diambil dari https://medium.com/@kholish219/tdd-dan-contoh-penerapannya-d164ea8c2b0b)

Proses kerja dalam TDD mengikuti siklus "*Red-Green-Refactor*"[21]. Siklus tersebut terdiri atas:

- *Red* (Merah)
  Pengembang menulis tes untuk fungsionalitas kecil (terisolasi) yang belum diimplementasikan. Karena kodenya belum ada, tes akan merah/gagal.
- *Green* (Hijau)
  Mengacu pada proses penulisan kode sederhana dengan tujuan membuat berhasil tes 'merah' sebelumnya. Pada tahap ini, kode dibuat hanya sampai tes 'pass' atau hijau.
- *Refactor*
  Proses pembersihan dan perbaikan struktur kode tes 'hijau', termasuk melengkapinya. Bertujuan menghilangkan duplikasi kode tanpa mengubah fungsionalitas yang sudah berhasil tersebut.

TDD memiliki keunggulan dan tantangan dalam penerapannya. Dari segi keunggulan, karena pengujian dilakukan secara terus-menerus selama pengembangan, cacat/*bug* dapat dideteksi dan ditangani lebih awal, pun dengan adanya refaktorisasi, menghasilkan desain lebih mudah diintegrasikan dengan fungsionalitas lain. Keunggulan lain ialah pengurangan kompleksitas karena pengembangan fungsionalitas dibagi dalam fungsionalitas kecil pada tiap iterasi [22]. Namun demikian, TDD memerlukan pemahaman yang baik dan keahlian dalam segi pengembangan perangkat lunak juga pengujiannya. Proses TDD dapat

memakan waktu lebih banyak, terutama ketika terjadi kegagalan berulang, yang mengakibatkan panjangnya perbaikan yang perlu dilakukan [22].

2.1.6. Moscow Prioritization

MoSCoW *prioritization* merujuk pada metode penentuan prioritas kebutuhan, dalam hal ini ialah prioritas dari *user stories* yang ada. Metode MoSCoW awalnya diterapkan pada metode pengembangan sistem DSDM (*Dynamic Systems Development Method*). Kata Moscow sendiri merupakan akronim dari *Must have, Should have, Could have, dan Won't have* [23]. Adapun keempatnya dijabarkan sebagai berikut.

Gambar 5. Moscow prioritization
(sumber: https://storiesonboard.com/blog/moscow-prioritization-model)

- *Must-have features*
  Fitur esensial yang harus ada dalam sistem, tidak boleh tidak ada.
- *Should-have features*
  Fitur penting dalam sistem yang sebaiknya ada, namun kesuksesan sistem tidak bergantung padanya. Apabila terdapat keterbatasan waktu, dapat ditinggalkan dalam versi rilis.
- *Could-have features*
  Fitur yang dapat ditinggalkan dalam versi rilis bila terdapat keterbatasan waktu tanpa ada dampak berarti pada sistem.
- *Won't have this time features*
  Fitur pengembangan yang diinginkan yang dapat dikembangkan pada versi rilis berikutnya.

2.1.7. Offline-first application

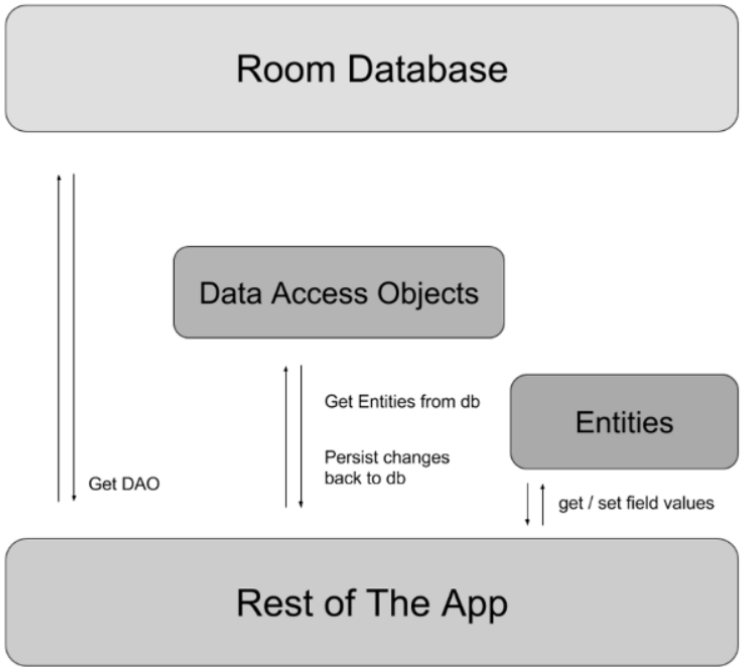
Pendekatan *offline-first* mengacu pada perancangan aplikasi yang berfungsi tanpa memerlukan koneksi internet. Berbeda dengan desain *cloud-first* dengan server menjadi otoritas tunggal, aplikasi *offline-first* menyimpan aksi secara lokal terlebih dahulu. Proses sinkronisasi dilakukan saat koneksi tersedia [24].

Pada aplikasi *Android*, basis data lokal menjadi *source of truth* bagi UI, dan aplikasi kembali diperbarui ketika mendapatkan jaringan. Adapun mekanisme *offline-first* pada android terbagi sebagai berikut [25].

- Pembacaan data
  UI mengambil data langsung dari Room. Di latar belakang, Repositori secara asinkron mengambil data terbaru dari server dan memperbarui *database* Room, yang kemudian otomatis memperbarui tampilan UI.
- Penulisan data
  Dilakukan penyimpanan perubahan dahulu ke Room (*optimistic update*). Kemudian didelegasikan ke *Background Task Scheduler*, seperti *WorkManager*, komponen yang memastikan data terkirim saat koneksi internet tersedia, bahkan jika aplikasi sempat ditutup.

Pendekatan *offline-first* dapat diimplementasikan dengan penggunaan pustaka Room (*Room persistence library*). *Library* ini berbasis SQLite yang memiliki anotasi guna mengurangi *boilerplate code* dan proses migrasi skema basis data sederhana. Room terdiri dari basis data, DAO (*Data Transfer Object*), dan *entity* [6].

- *Entity* mengacu pada kelas data (*data class*) representasi tabel suatu basis data. Setiap properti dalam kelas Entity mewakili sebuah kolom pada tabel tersebut.
- *Data Access Object* (DAO) ialah antarmuka untuk mengakses basis data, termasuk ke dalam tabel. Berupa jembatan aplikasi dan basis data, berisi *query* SQL untuk melakukan operasi (menambah, membaca, memperbarui, dan menghapus data (CRUD)).



Gambar 6. Struktur pustaka Room [6]

- Dalam pembacaan data, aplikasi memanggil sebuah metode pada DAO. DAO akan mengeksekusi *query* dan mengambil data tersebut dari basis data. Kemudian akan disimpan dalam bentuk objek atau daftar objek *Entity*. *Entity* ini kemudian diakses aplikasi untuk menampilkan data.
- Dalam penyimpanan/pembaruan data, aplikasi mengakses (buat/modifikasi) *Entity*. Objek diteruskan sebagai parameter ke metode pada DAO. DAO melaksanakan *queri* (tulis, hapus, tambah, dan lainnya) pada basis data.

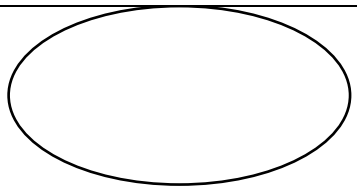
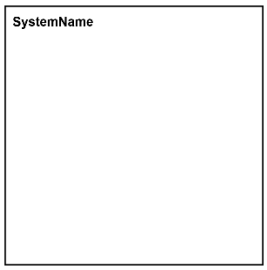
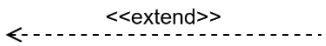
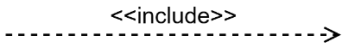
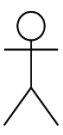
2.1.8. Unified Modeling Language

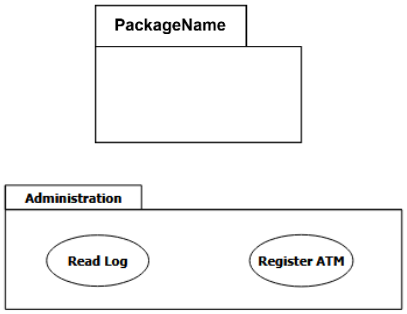
Unified Modeling Language (UML) adalah bahasa pemodelan visual untuk memvisualisasikan, membangun, dan mendokumentasikan sistem perangkat lunak yang digagas oleh OMG (*Object Management Group*) pada 1997. Terdapat beberapa diagram, termasuk dalamnya *activity diagram, class diagram, communication diagram, deployment diagram, state machine diagram, sequence diagram, use case diagram,* dan lain lain.

2.1.8.1.Use Case Diagram

Use Case digunakan untuk memetakan kebutuhan sistem, terkhusus hal yang harus dilakukan oleh sistem. Terdapat berbagai simbol, yakni *Actor, Use Case*, dan *subject*. *Subject* merepresentasikan sistem, sedangkan pengguna atau sistem lain yang berinteraksi dengan *subject* direpresentasikan sebagai *Actor*. Notasi standar dalam *Use Case Diagram*[26] tertera pada tabel berikut.

Tabel 1. Notasi *Use Case Diagram*

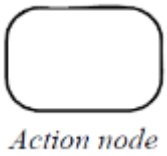
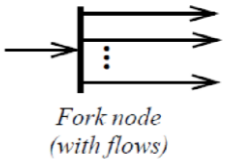
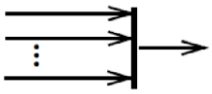
Notasi	**Deskripsi**
	***Use Cases*** Berbentuk elips yang berisi nama *use case* (di dalamnya/ bawah)
	***System Boundary*** Persegi panjang dengan nama di sudut kiri atas, *use case* berada di dalamnya, diguanakn untuk menunjukkan batas sistem.
	***Extend Relationship*** Hubungan opsional, digambarkan dengan panah putus-putus berkepala terbuka dari *use case* pemberi ekstensi ke *use case* yang diperluas, disertai label «extend».
	***Include relationship*** Penyertaan fungsionalitas, digambarkan dengan panah putus-putus berkepala terbuka dari *use case* dasar ke *use case* yang disertakan, disertai label «include».
	***Actor*** ikon manusia yang menggambarkan pengguna sistem dengan nama aktor berada di dekat ikon tersebut.

 Figure 18.11 Example UseCases owned by Packages	**Packages** Notasi dasar UML, untuk pengorganisasian/pengelompokan model yang memiliki kesamaan.

2.1.8.2.Activity Diagram

Activity diagram adalah jenis diagram yang menggambarkan perilaku (*Behavior*) sebagai grafik *node* yang saling terhubung oleh tepi (*edges*). Notasi standar dalam *Activity Diagram*[26] tertera pada tabel berikut.

Tabel 2. Notasi *Activity Diagram*

Notasi	**Deskripsi**
	***Action Node*** Menggambarkan langkah eksekusi suatu aktivitas, terjadi ketika suatu kondisi terjadi.
	**Initial Node** *starting point*/Titik awal aktivitas.
	***Final Node*** *finish point*/Titik akhir aktivitas.
	***Fork Nodes*** Memecah satu aliran menjadi beberapa aliran yang berjalan.
	***Join Nodes*** Sinkronisasi/ penggabungan kembali beberapa aliran menjadi satu aliran keluar.

	***Decision Nodes*** Percabangan atau pemilihan aliran keluar berdasarkan kondisi tertentu.
	***Connector*** Penghubung alur antar *activity edge*. Tiap konektor harus memiliki pasangan yang sama persis, berarti alur tersebut berlanjut.
 *Regular activity edge*	***Activity Edge/ ControlFlow/ ObjectFlow*** Koneksi antar *node* aktivitas, baik berupa *Control Flow* maupun *Object Flow*, dinotasikan dengan garis panah terbuka.

2.1.9. Unit testing

Dalam sebuah pengujian, target pengujian bervariasi tergantung SUT (*System Under Test*), kondisi lingkungan, waktu, dan lainnya. Salah satunya ialah tingkat unit/ *unit testing*. *Unit testing* memverifikasi fungsionalitas elemen-elemen SUT yang diuji secara terpisah dan terisolasi. Umumnya, pengujian unit dilakukan oleh pengembang yang menulis kode tersebut [27]. *Unit testing* mengacu pada sebuah pengujian perangkat lunak otomatis yang melakukan verifikasi pada bagian kecil/unit kode, berjalan cepat, dan dilakukan secara terisolasi [28]. Pun *unit testing* termasuk dalam praktik TDD dengan *test* ditulis di awal sebelum membentuk suatu kode.

Unit testing bertujuan mengelola keberlanjutan (*sustainable growth*) proyek [28]. Dengan adanya pengujian ini, fungsionalitas yang ada akan tetap berjalan bahkan ketika dilakukan penambahan atau refaktor kode untuk keperluan pengembangan aplikasi di masa depan. Tanpa pengujian, sebuah proyek akan mengalami *software entropy*, di mana kualitas kode menurun seiring waktu, yang menyebabkan kecepatan pengembangan melambat [28].

2.1.10. Software Maintainability

Maintainability merujuk pada kemudahan sebuah program diperbaiki apabila ditemukan *error*, perubahan *environment*, atau perubahan kebutuhan pengguna/*requirement changes* [18]. Suatu sistem harus dapat berkembang dan tetap mampu memenuhi kebutuhan pengguna di masa mendatang [29]. Adapun subkategori dalam *maintainability* menurut standar ISO/IEC 25010:2023 meliputi *modularity, reusability, analysability, modifiability,* dan *testability* [30].

Terdapat beberapa metrik yang dapat digunakan, diantaranya *maintainability indeks, cyclomatic complexity,* serta *halstead suite.*[31]. Selain itu, terdapat metrik yang tidak secara langsung berhubungan dengan tingkat *maintainability* suatu sistem namun berpengaruh dalam pengembangan kode, yakni *code coverage*.

2.1.10.1. Maintainability Index

Guna mengukur *maintainability* suatu sistem, terdapat metrik yakni *Maintainability Index*. Metrik ini diusulkan oleh Oman dan Hagemeister. Pengukuran ini mempertimbangkan metrik yang merefleksikan *mental effort* yang diperlukan untuk memelihara *software*, kompleksitas kode, dan ukuran kode. Adapun pengukurannya dilakukan berdasarkan rumus berikut.

$$MI = 171 - 5.2 \times \ln(aveV) - 0.23 \times aveV(g) - 16.2\, x \ln(aveLOC) \qquad (2)$$

Keterangan:

- $aveV$ = rata-rata Halstead's Volume per module
- $aveV(g)$ = rata-rata Cyclomatic Complexity per module
- $aveLOC$ = rata-rata Lines of Code per module

Terdapat interpretasi berdasarkan nilai yang diperoleh, yakni [32]

- Nilai di bawah 65 sebagai *low maintainability*
- Nilai antara 65 dan 85 sebagai *medium maintainability*
- Nilai di atas 85 sebagai *high maintainability*

2.1.10.2. Cyclomatic Complexity

Cyclomatic Complexity ialah salah satu metrik pengukuran *maintainability,* yakni *modularity* dan *testability* [33], terkhusus guna mengukur kompleksitas logika. Pertama kali dikembangkan oleh Thomas J. McCabe pada tahun 1976 guna identifikasi tingkat kerumitan suatu program [34]. Dalam pengujian *basis path*, nilai yang dihitung dari *Cyclomatic Complexity* mendefinisikan jumlah jalur independen dalam program [18].

McCabe menggambarkan perhitungan *Cyclomatic Complexity* sebagai graf. Pun dipaparkan bahwa ambang batas skor *cyclomatic complexity* yang baik berada di bawah 10 [18], [34]. Adapun cara untuk mengukur *Cyclomatic Complexity* suatu fungsi dipaparkan sebagai berikut [34].

1. Teori Graf (*The Graph Theory Formula*)

$$v(G) = e - n + 2p \quad (3)$$

Dapat disederhanakan sebagai berikut karena nilai p = 1 untuk fungsi tunggal

$$v(G) = e - n + 2 \quad (4)$$

Keterangan:

$v(G)$: *Cyclomatic Complexity*

e : Jumlah *edges*/sisi

n : Jumlah *nodes*/simpul

p : Jumlah *connected components*/komponen terhubung, pada satu fungsi atau metode tunggal, nilai p hampir selalu 1.

2. Wilayah Graf (*Counting Regions*)

Cyclomatic Complexity dapat pula dihitung berdasarkan jumlah wilayah graf nya. Setiap kali sebuah alur membelah dan menyatu kembali, tercipta wilayah tertutup baru.

3. Titik Keputusan (*Counting Predicate Nodes*)

$$v(G) = \pi + 1 \quad (5)$$

Keterangan:

$v(G)$: *Cyclomatic Complexity*

π : Jumlah *predicate nodes* /titik keputusan kode

Adapun titik keputusan mengacu pada ekspresi yang dapat menghasilkan lebih dari satu kemungkinan hasil (true/false). Dalam pemrograman contohnya penggunan logika percabangan dan perulangan. Terdapat beberapa alat bantu yang dapat digunakan untuk menghitung skor, salah satunya ialah SonarQube dan Detekt.

2.1.10.3. Cognitive Complexity

Cognitive Complexity adalah metrik yang dirancang untuk mengatasi kelemahan *Cyclomatic Complexity* dalam mengukur *maintainability*. *Cyclomatic Complexity* unggul dalam mengukur keterujian (*testability*), namun tidak menjangkau kesulitan pemahaman kode [35]. *Cognitive Complexity* mengukur kesulitan relatif dalam pemahaman, termasuk pemeliharaan *method, class*, dan *applications*.

Terdapat beberapa prinsip dalamnya, dijabarkan sebagai berikut.

1. Abaikan struktur yang menyederhanakan penulisan kode.
2. Tambahkan satu poin (*increment*) untuk setiap pemutusan alur linear kode.
3. Tambahkan poin ketika struktur bersarang (*nested*).

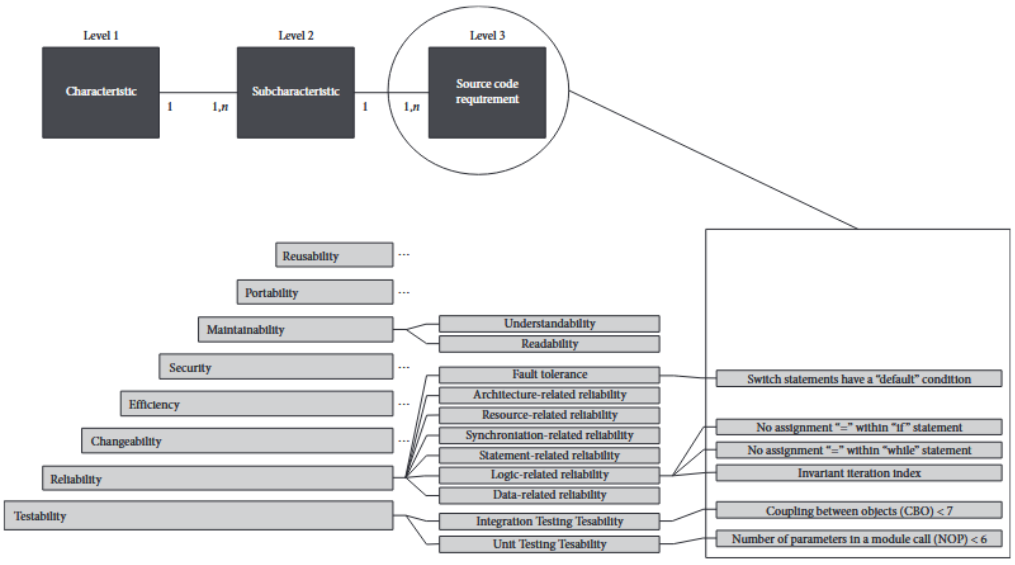
Campbell dalam *white paper Cognitive Complexity* tidak menetapkan nilai ambang batas secara eksplisit. Terdapat beberapa alat bantu yang dapat digunakan untuk menghitung skor *Cognitive Complexity*, salah satunya ialah SonarQube dan Detekt. Sesuai dengan dokumentasi resmi *detekt*, aturan *CognitiveComplexMethod* memiliki nilai ambang batas sebesar 15 [36].

2.1.10.4. Code Coverage

Code coverage dapat mengukur seberapa menyeluruh suatu pengujian dilakukan. Metrik ini menunjukkan rasio baris kode yang dieksekusi oleh *test* dengan total jumlah baris dalam *source code* [28]. Skor dengan persentase tinggi, dalam hal ini semakin mendekati 100%, berarti pernyataan yang dapat dieksekusi di kode/*executable code* telah dijalankan [37]. Hal ini berarti apabila ada pernyataan yang memiliki *defect*, pernyataan tersebut dieksekusi dan ketika tidak berhasil berarti menunjukkan lokasi *defect* tersebut. Metrik ini tidak secara langsung berhubungan dengan tingkat *maintainability* suatu sistem namun berpengaruh dalam pengembangan kode yang ada.

2.1.10.5. Software Quality Enhancement (SQALE)

Software Quality Enhancement (SQALE) ialah metrik pengukuran kualitas software berdasarkan standar ISO 9126 (sekarang telah digantikan oleh ISO 25010). Pada metrik ini, terdapat 3 level, tergambar sebagai piramida [38].



Gambar 7. Pembagian level pada metriks terkait sofware quality[38]

Gambar 7 menunjukkan konsep level dalam SQALE. Level 1 disebut *characteristic*, mencangkup *reusability, portability, testability*, dan lainnya. Masing-masing karakteristik dapat memiliki sub-karakteristik (level 2), contohnya terdapat *integration test testability* dan *unit test testability*. Pada tiap sub-karateristik dijabarkan lagi menjadi *source code requirement* (level 3). *Source code*

requirement inilah yang menjadi aturan teknis *technical debt* (TD). TD ialah jumlah *remediation cost* yang diperlukan untuk menuntaskan pelanggaran *rule*. Adapun rumus perhitungan *remediation cost* ialah sebagai berikut.

$$RC = \frac{\sum_{rule} effortToFix(violations_{rule})}{8\ [\frac{hr}{day}]} \quad (6)$$

Keterangan:

- Effort to Fix = Estimasi waktu (menit) untuk memperbaiki pelanggaran
- $\sum_{rule}$ = jumlah estimasi *effort* dari tiap rule

Remediation Cost (RC) tersebut merepresentasikan *cost* yang diperlukan untuk memperbaiki pelanggaran *rule* pada tiap kategori (*testability, reliability*, *security,* dan lainnya).

2.1.10.6. Maintainability Rating

Maintainability rating ialah penilaian *maintainability* yang dilakukan dengan *tools* SonarQube, ditunjukkan dalam bentuk rasio kepadatan *technical debt* terhadap ukuran kode [39] yang diinterpretasikan dalam rentang berikut (semakin rendah rasionya, semakin baik kodenya).

- **A** ≤ 5% hingga 0%
- **B** ≥ 5% hingga <10%
- **C** ≥ 10% hingga <20%
- **D** ≥ 20% hingga < 50%
- **E** ≥ 50%

Guna mempermudah pengukuran metrik-metrik tersebut, terdapat *tools* pengukuran otomatis, diantaranya ialah CoverageRunner, Detekt, dan SonarQube. CoverageRunner dapat mengukur *code coverage* dari *script* uji yang ada. Alat bantu *Detekt* mengukur beberapa metrik, diantaranya *cyclomatic complexity, cognitive complexity, line of codes*, dan lainnya, namun tidak memberikan skor *maintainability* secara spesifik.

SonarQube disebut sebagai salah satu *tools* yang umum digunakan dalam pengukuran metrik *maintainability* [31]. SonarQube mengukur *maintainability* dengan *technical debt*, yakni jumlah *remediation cost* yang didapatkan dari *maintainability issue* [39]. SonarQube awalnya menghitung *technical debt* menggunakan metode *Software Quality Enhancement* (SQALE). Namun seiring berjalannya waktu, *Reliability* and *Security issues* dipisahkan menjadi kategori tersendiri dan perhitungan SQALE sisanya diubah nama menjadi *maintainability rating* [40].

2.1.11. Black box testing

Salah satu metode utama dalam pengujian fungsional ialah *Black Box Testing*. Metode ini menguji fungsionalitas sistem tanpa adanya analisis struktur kode internal [28]. Didasarkan pada spesifikasi, bukan implementasinya dalam kode sumber atau perangkat lunak yang dapat dieksekusi [41].

Kelebihan utama dari *black box testing* adalah menguji fungsionalitas sistem dan fitur yang diperlukan oleh pengguna. Namun, kekurangannya adalah pengujian ini hanya mencakup permukaan aplikasi dan tidak dapat mendeteksi kesalahan pada level kode.

2.1.12. User Acceptance Test (UAT)

Acceptance testing ialah tahap penting dalam pengembangan perangkat lunak dengan fokus pada validasi dan demonstrasi kesiapan *deployment* sebuah sistem [37]. Bertujuan melakukan verifikasi bahwa sistem, tepatnya SUT (*System Under Test*) yang dibangun memenuhi persyaratan kebutuhan pengguna akhir [27]. Adapun pengujian ini dilakukan secara langsung bersama *end user*, pengguna akan menjalankan fungsi dan tugas sesuai tujuan pengembangan perangkat lunak [27], [37].

Salah satu bentuk dari *acceptance testing* ialah *User Acceptance Testing*, merujuk pada tahap akhir pengujian perangkan lunak dengan pengguna yang diberikan kesempatan berinteraksi dengan sistem sebelum diluncurkan. Dilakukan dengan pengguna akhir yang menguji sistem dan memutuskan diterima atau tidaknya sistem [29].

Terdapat beberapa bentuk *acceptance testing*, diantaranya *alpha testing* dan *beta testing* [18]. *Alpha testing* merujuk pada pengujian yang dilakukan di lokasi pengembang oleh sekelompok perwakilan *end user*. Dilakukan secara terkontrol, pengembang mengamati dan mencatat kesalahan atau masalah penggunaan yang dialami oleh *end user*. Pengujian dapat dilakukan dengan *black box testing*. Sementara *Beta testing* merujuk pada pengujian yang dilakukan berdasarkan *end user*. merupakan pengujian langsung penggunaan aplikasi oleh *end user* sesuai dengan lingkungan nyata tidak terkontrol yang sebenarnya.

2.2.Penelitian Terkait

Penelitian yang dilakukan oleh Gloria Ejehiohen Iyawa (2020) yang berjudul ”*Personal Extreme Programming: Exploring Developers' Adoption*”. Penelitian ini meneliti penerapan metode PXP dalam pengembangan suatu aplikasi secara individu. Penulis menggunakan teori *Diffusion of Innovation* (DoI) dan *Technology Acceptance Model* (TAM) untuk mengidentifikasi faktor-faktor yang memengaruhi pengembang perangkat lunak dalam mengadopsi PXP. Pengujian dilakukan dengan wawancara pada sepuluh pengembang yang menggunakan PXP, didapatkan hasil bahwasanya dengan penerapan PXP meningkatkan efisiensi kerja untuk proyek berskala kecil dan mempercepat pemahaman akan *user stories* [19].

Penelitian yang dilakukan oleh Marthasari (2018) berjudul “*Personal Extreme Programming with MoSCoW Prioritization for Developing Library Information System*”. Penelitian ini mengembangkan sebuah aplikasi *Batu State Attorney library applications* dengan metode pengembangan *Personal Extreme Programming* (PXP). Pada tahap *planning*, peneliti menggunakan metode Moscow

sebagai alat bantu penentuan prioritas *user stories* yang ada. Adapun pengujian dilakukan oleh pengguna akhir melalui UAT, dengan melakukan pengujian bersama *client* berdasarkan *user story* yang telah dibuat [42].

Penelitian yang dilakukan oleh Musofa (2025) berjudul "Implementasi Metode Personal Extreme Programing (PXP) pada pengembangan Aplikasi Buku Tamu (Studi Kasus: Dinas Komunikasi dan Informatika Kabupaten Lombok Tengah)". Penelitian ini mengembangkan buku tamu digital berbasis website. Peneliti menerapkan metode PXP mencangkup fase *requirement, planning, iteration initialization, design, implementation, system testing,* dan *retrospective*. Aplikasi dikembangkan dengan PHP *framework* Laravel dan *database* MySQL dan menerapkan pemindaian QR *code* untuk registrasi otomatis. Pengujian fungsional menggunakan *black box testing* dengan keberhasilan fungsional 100% dan akurasi verifikasi identitas tamu 98,7% [43].

Penelitian yang dilakukan oleh Ren dan Barrett (2023) berjudul "*Test-driven development, engagement in activity, and maintainability: An empirical study"*. Penelitian ini melibatkan 237 mahasiswa untuk mencari hubungan praktik TDD, keterlibatan dalam pengembangan, serta dampaknya terhadap tingkat keterpeliharaan (*maintainability*) suatu perangkat lunak. Penelitian menggunakan analisis regresi *Ordinari Least Squares (OLS)* untuk mengukur dampak aktivitas terhadap kualitas kode. *Cyclomatic Complexity* (CC) digunakan sebagai salah satu pengukuran. Didapatkan bahwa penerapan pendekatan TDD pada pengembangan suatu perangkat lunak menghasilkan skor CC yang rendah, yang berarti meningkatkan *maintainability* [44].

Penelitian yang dilakukan oleh Papis, dkk (2022) berjudul "*Experimental evaluation of test-driven development with interns working on a real industrial project*". Penelitian melibatkan 19 peserta dalam mengevaluasi pendekatan *Test Driven Development* (TDD) dan *Test-Last Development* (TLD). Peneliti mengukur jumlah *bug* dan kualitas pengujian melalui *code coverage*. Didapatkan hasil bahwa pembuatan *unit testing* menghasilkan kode yang lebih rendah *bug* dibanding tanpa

unit testing. Didapatkan juga bahwa pengembangan dengan pendekatan TDD menghasilkan *bug* per iterasi 1.8x lebih rendah dibanding TLD dan kualitas tes 5% lebih tinggi dibanding TLD dihitung dari *code coverage* [45].

Penelitian yang dilakukan oleh Raty (2025) berjudul "*Clean Architecture Android Application for Surveying HVAC System"*. Penelitian ini mengembangkan aplikasi Android dengan arsitektur *Clean Architecture* pada sebuah sistem HVAC. Aplikasi yang dikembangkan ialah SurveyApp, pendukung teknisi lapangan untuk melakukan survei dan pendokumentasian sistem HVAC (*Heating, Ventilation, and Air Conditioning*) yang dibuat agar bisa bekerja tanpa internet. Peneliti menerapkan pemisahan logika aplikasi ke dalam tiga lapisan utama, yaitu *presentation layer*, *domain layer*, dan *data layer*. Peneliti juga menggunakan *Work Manager* untuk sinkronisasi data latar belakang. Aplikasi dibangun pada sistem operasi android dengan bahasa pemrograman Kotlin dan *framework* Jetpack Compose. Peneliti tidak membahas secara eksplisit dampak penggunaan arsitektur terhadap aplikasi, namun didapatkan hasil bahwa aplikasi yang dibangun membantu pencatatan dan pendokumentasian proses survei dan meningkatkan kecepatan pendokumentasiannya [46].

Penelitian yang dilakukan oleh Gaffney dkk (2022) berjudul "*SQLite: Past, Present, and Future*". Penelitian ini mengevaluasi performa SQLite sebagai *database*. Analisis dilakukan dengan membandingkan performa SQLite dengan *database* lain dengan *Online Analytical Processing (*OLAP). Didapatkan hasil bahwasanya SQLite memiliki peforma kecepatan pemrosesan 4,2x lebih cepat (dengan pengukuran *Star Schema Benchmark*) [47].

Penelitian yang dilakukan oleh Pothineni (2024) berjudul "*Offline-First Mobile Architecture: Enhancing Usability and Resilience in Mobile Systems*". Pothineni melakukan penelitian terkait arsitektur *offline-first* aplikasi mobile agar tetap berfungsi dengan baik dalam kondisi *offline*, termasuk strategi sinkronisasi, *best practice* industri, dan lainnya. Penelitian memaparkan terdapat beberapa logika sinkronisasi, yakni *Conflict-Resolution Data Types* (CRDT) untuk aplikasi

kolaboratif yang diedit oleh banyak pengguna bersamaan (figma, automerge), *Operational Transformation* (OT) sinkronisasi terkait dokumen terstruktur (*code editing, spreadsheet management*), dan *Last-Write-Wins* (LWW) yang melakukan *owerwrite* berdasarkan waktu. Terdapat konsep *delta logging*, berarti pencatatan perubahan spesifik data, bukan mengirim keseluruhan data berkali-kali. Setiap pendekatan memiliki karakteristik berbeda dan pemilihan pendekatan bergantung pada kebutuhan yang ada [24].

Penelitian yang dilakukan oleh Ardito (2020) berjudul "*A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review*". Peneliti melakukan *literature review* pada 174 metrik *software maintainability* dari sumber ACM *Digital Library*, IEEE Xplore, Scopus, dan Web of Science. Peneliti kemudian mengambil 15 metrik paling populer yang digunakan untuk mengukur *maintainablity* sistem (dalam berbagai bahasa pemrograman). Metrik tersebut diurutkan alfabetikal dan diantaranya terdapat *maintainability indeks* (MI), *halstead volume* (HV), *McCabe cyclomatic complexity* (MCC), dan lainnya. Peneliti juga memaparkan *tools* yang dapat digunakan, contohnya penggunaan SonarQube pada sistem dengan bahasa pemrograman Kotlin [31].

Penelitian yang dilakukan oleh Strečanský dkk. (2020) berjudul "*Comparing Maintainability Index, SIG Method, and SQALE for Technical Debt Identification*". Peneliti menggunakan metode *Maintainability Index* (MI), SIG *maintainability model*, dan analisis SQALE untuk menganalisis 20 *open source Python libraries* pada tiap versi rilisnya secara *time series* dan membandingkan hasil *report* ketiga metode. SQALE cenderung memberikan *report* lebih stabil antar versi rilis, sementara MI dan SIG TD lebih terlihat perubahan/pertumbuhan perbaikannya di tiap versi rilis. Walau menggunakan metrik/rumus berbeda, ketiganya dapat digunakan untuk mengukur *maintainability* sistem dan bahkan dapat digunakan sebagai kombinasi pengukuran [38].

III. METODE PENELITIAN

3.1. Waktu dan Tempat

3.1.1. Waktu Penelitian

Penelitian ini akan dilaksanakan selama 6 (enam) bulan, Tabel berikut menjelaskan waktu pelaksanaan penelitian.

Tabel 3. Waktu Pelaksanaan Penelitian

No	Kegiatan Penelitian	Nov	Des	Jan	Feb	Mar	Apr
1	Studi Literatur						
2	*Requirements*						
3	*Planning*						
4	*Iteration initialization*						
5	*Design*						
6	*Implementation*						
7	*System Testing*						
8	*Retrospective*						
9	Pengujian pengguna						
10	Perbaikan						
11	Penyerahan proyek						
12	Penulisan laporan						

3.1.2. Tempat Penelitian

Penelitian ini dilaksanakan di Dinas Kehutanan Provinsi Lampung, yang berlokasi pada Jalan Zaenal Abidin Pagar Alam, Rajabasa, Kec. Rajabasa, Kota Bandar Lampung, Lampung 35141.

3.2. Alat dan Bahan Penelitian

3.2.1. Alat Penelitian

Adapun alat yang digunakan dalam penelitian dibagi menjadi dua jenis, yaitu perangkat keras dan perangkat lunak seperti ditunjukkan oleh tabel berikut.

Tabel 4. Perangkat Keras

No	Perangkat Keras	Spesifikasi	Kegunaan
1	Laptop	intel i7-1165G7, RAM 20GB, Windows 11 pro	Pengembangan aplikasi dan penulisan laporan penelitian.
2	*Smartphone*	Android 10 SDK 29	Pengujian aplikasi

Tabel 5. Perangkat Lunak

No	Perangkat Lunak	Versi	Keterangan
1	Android Studio IDE 	Koala *Feature* Drop 2024.1.2	Editor kode untuk implementasi program dan pengujian aplikasi
2	Figma 	Versi Web	Alat pengembangan desain
3	Detekt 	1.23.6	Alat bantu analisis kode
4	IntelliJ IDEA Coverage Runner	*Bundled* Android Studio Koala	Alat bantu pengukuran *code coverage*
5	SonarQube 	5.0.0.4638	Alat bantu pengukuran *maintainability*

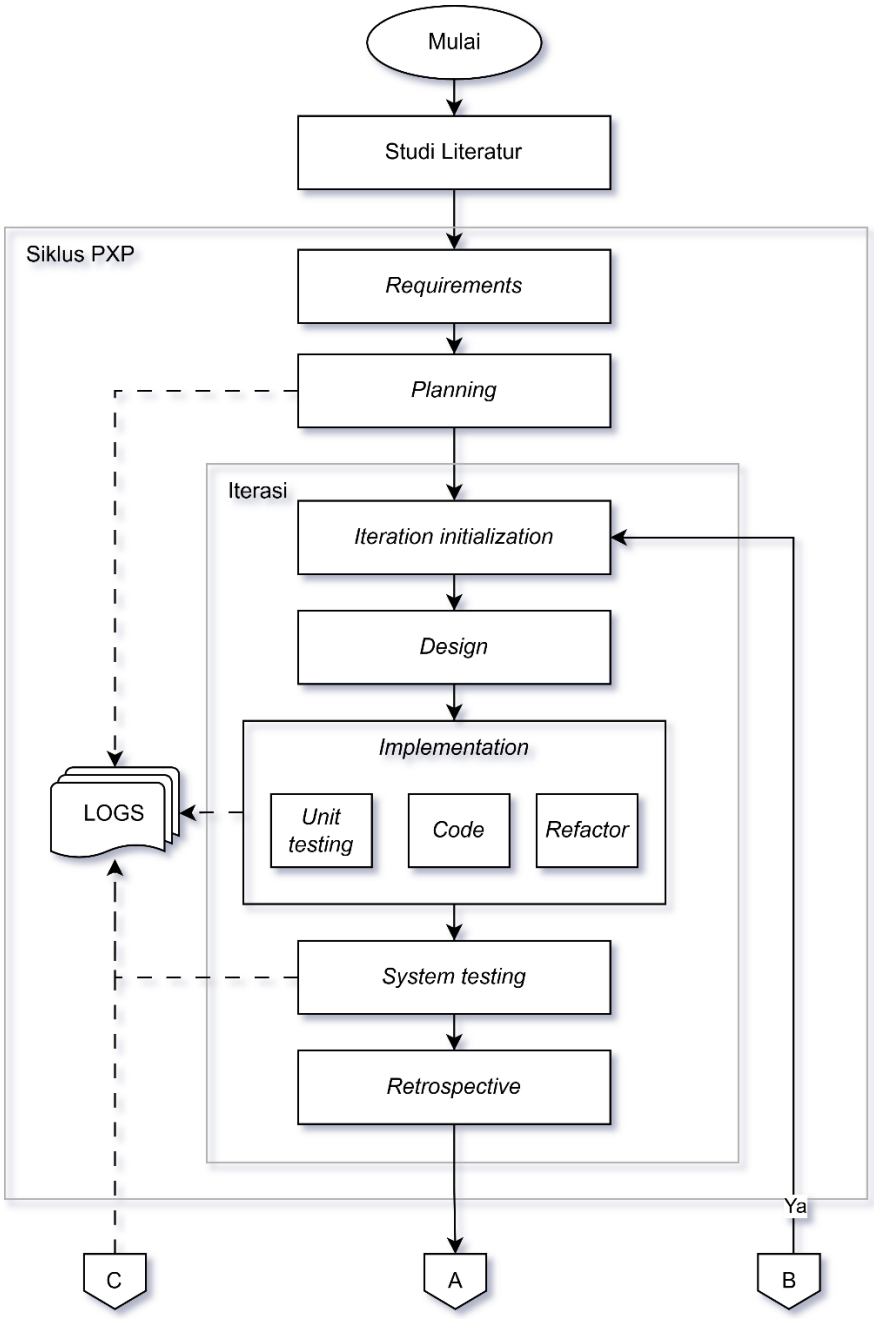
3.2.2. Bahan Penelitian

Penelitian ini memanfaatkan beberapa sumber sebagai bahan referensi untuk memperkuat pemahaman, teknik, dan langkah-langkah dalam pengembangan aplikasi, diantaranya adalah sebagai berikut.

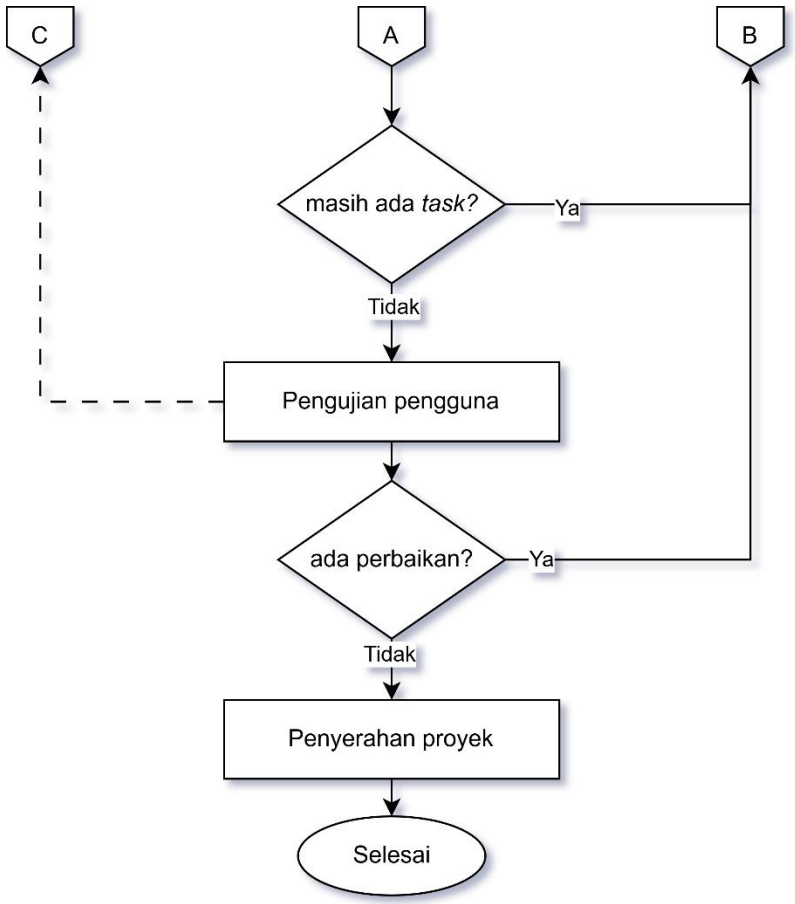
1. Penelitian terdahulu yang relevan.
2. Artikel dan jurnal internasional dan nasional sebagai rujukan penelitian.
3. Dokumentasi dari situs resmi teknologi yang digunakan.
4. *Website* Sitanihut Dinas Kehutanan Provinsi Lampung yang telah dikembangkan.
5. API Sitanihut yang telah dikembangkan.

3.3. Tahapan Penelitian

Metode pengembangan perangkat lunak yang digunakan ialah *Personal Extremme Programming* (PXP). PXP terdiri atas fase *requirement*, *planning*, *iteration initialization*, *design*, *implementation*, *system testing*, dan *retrospective*. Tahapan dalam penelitian ini ditunjukkan oleh gambar 7 sebagai berikut.



Gambar 8. Tahapan Penelitian



Gambar 7. Tahapan Penelitian (lanjutan)

Pada gambar 7 dipaparkan tahapan penelitian dalam pengembangan aplikasi Sitanihut. Dimulai dengan tahap Studi Literatur sebagai peninjauan pustaka dan landasan penelitian awal. Kemudian dilaksanakan pengembangan aplikasi dengan metode PXP, dimulai dari tahap *requirements*, *planning,* memasuki iterasi dengan fase *iteration initialization*, *design, implementation* (mencangkup *unit test, code*, dan *refactor*). Pada tahap ini, dilakukan penyimpanan *logs* atau catatan selama pengembangan. Kemudian dilakukan *system testing* pada fitur fitur yang diimplementasikan pada iterasi tersebut. Temuan-temuan maupun *defects* yang ditemukan pada pengujian ini dicatat dalam *logs* dan menjadi *task* atau tugas pada iterasi selanjutnya sesuai prioritas. Iterasi selanjutnya akan mengeksekusi daftar tugas dari tahap perencanaan/*planning* dan *task* dari *logs*. Setelah seluruh *task* dikerjakan, dilakukan pengujian pada pengguna akhir. Berdasarkan pengujian tersebut, didapatkan umpan balik dan dilakukan perbaikan. Setelah perbaikan selesai dilakukan, maka proyek diserahkan pada instansi.

3.3.1. Studi Literatur

Studi literatur digunakan dalam peninjauan pustaka, metodologi, penelitian yang relevan, pemilihan teknologi yang digunakan, serta riset teoritis. Studi literatur penelitian ini digunakan untuk mengumpulkan informasi terkait sebagai bahan referensi dan mendukung proses pelaksanaan penelitian. Informasi yang dikumpulkan mencangkup artikel, jurnal, dan dokumentasi yang kredibel dan relevan untuk dijadikan referensi dalam pengembangan sistem.

3.3.2. Requirement

Membuat daftar kebutuhan fungsional dan non-fungsional terkait sistem yang akan dikembangkan. Kebutuhan tersebut disusun berdasarkan sistem Sitanihut yang telah dikembangkan dan kebutuhan pengguna dari hasil pengumpulan data. Adapun pengumpulan data dilakukan dengan wawancara. Berdasarkan hasil pengumpulan kebutuhan tersebut kemudian dibuat *user stories*.

3.3.3. Planning

Tahap *planning* berisi penyusunan *task*/tugas yang akan dikerjakan selama penelitian berdasarkan hasil tahap *requirement* sebelumnya. Pada tahap ini, *user stories* yang telah dibuat disusun prioritasnya. Penyusunan prioritas *user stories* ini dilakukan dengan menggunakan metode Moscow. Juga dilakukan penentuan *story point* guna mengukur estimasi waktu pengerjaan (dalam hari) tiap *story* dan akumulasinya di tiap iterasi.

3.3.4. Iterasi

3.3.4.1. Iteration Initialization

Tahap *iteration initialization* menandakan awal/*start* dari setiap iterasi pengembangan. Pada tahap *iteration initialization* ini, dilakukan pengambilan *user stories* yang telah disusun prioritasnya pada tahap sebelumnya yang akan dilaksanakan pada iterasi tersebut.

3.3.4.2.Design

Pada tahap ini dilakukan perencanaan desain dari sistem yang akan dibuat dalam proses pengembangan proyek dengan metode PXP. Dilakukan perencanaan desain *high fidelity* antarmuka pengguna, termasuk tata letak, penggunaan komponen, tipografi, margin, dan sebagainya untuk kemudian diimplementasikan pada tahap selanjutnya. Pada tahap ini, dilakukan juga pemodelan struktur *local database* aplikasi dengan *Entity Relationship Diagram* (ERD) serta diagram aktivitas yang dilakukan *user* dalam *activity diagram*. Perencanaan desain ini dilakukan guna meminimalisir waktu dalam proses pengembangan tampilan pada tahap selanjutnya.

3.3.4.3.Implementation

Pada tahap ini, dilakukan implementasi pengembangan aplikasi berdasarkan desain yang telah dibuat sebelumnya. Adapun hal yang dilakukan termasuk penyusunan pengujian kode dengan implementasi *unit testing*, pengembangan kode program, kemudian proses refaktor untuk pembersihan dan pelengkapan kode. Dalam pengembangan kode program, digunakan *tools* sebagai alat bantu pengembangan, yakni Android Studio sebagai IDE utama pengembangan aplikasi. Adapun aplikasi dikembangkan dengan bahasa Kotlin, yang pembangunan antarmuka dari desain ke kode dengan Jetpack Compose.

Arsitektur aplikasi yang digunakan ialah *Clean Architecture* guna *maintainability* di masa mendatang serta *testability* yang baik dengan adanya pemisahan *layer* yang dalam implementasinya diadaptasi menjadi model 3 lapis, yakni *domain, data,* dan *presentation* dengan dependensi utama mengarah ke layer domain. Untuk kebutuhan *offline*, digunakan basis data lokal dengan Room/SQLite.

3.3.4.4. System testing

Tahap pengujian sistem terhadap keseluruhan fitur yang telah dikembangkan. Pengujian ini dilakukan untuk memastikan tiap fitur berjalan sesuai fungsionalnya yang telah didefinisikan. Adapun pengujian yang dilakukan ialah pengujian *black box testing* dari pihak pengembang untuk fitur aplikasi yang ada. Pada iterasi

terakhir, dilakukan pengujian metrik-metrik *maintainability,* diantaranya ialah *maintainability index*, *cyclomatic complexity,* dan *code coverage* terhadap seluruh fitur yang telah dikembangkan. *Maintainability index* dan *cyclomatic complexity* dipilih sebagai top 15 metrik *maintainability* yang populer menurut *literature review*. Pun *cyclomatic complexity* termasuk dalam metrik pengukuran *modularity* dan *testability*. Sementara pengukuran *code coverage* digunakan untuk mengecek cakupan kode *unit test* yang telah dikembangkan selama fase TDD. Selain itu digunakan *tools* SonarQube yang mengimplementasikan metode SQALE untuk menghitung *maintainability rating*. Adapun *tools* ini termasuk *tools* populer dalam analisis kode, terutama pada bahasa Kotlin dan Jawa.

3.3.4.5. Retrospective

Dilakukan analisis terhadap data yang terkumpul selama fase-fase sebelumnya. Pengembang kemudian mengukur waktu nyata yang berjalan dalam tiap iterasi dan membandingkannya dengan estimasi waktu pengerjaan yang didefinisikan sebelumnya.

3.3.5. Pengujian pengguna

Guna menguji kesesuaian pengembangan sistem dengan kebutuhan pengguna, dilakukan *acceptance test* berupa *User Acceptance Test* dengan metode *alpha testing* yang ditujukan pada *end-user* menggunakan *black box.* Pengujian ini dilakukan setelah seluruh fitur selesai dan tidak ada lagi daftar *task* yang perlu dilakukan. Pengujian dilakukan pada pengguna akhir/*end user* meliputi penyuluh kehutanan, petani hutan, serta penanggung jawab dari pihak Dinas Kehutanan Provinsi Lampung yang akan bertanggung jawab atas verifikasi pelaporan petani. Berdasarkan pengujian tersebut, pengembang kemudian melakukan perbaikan pada fitur yang membutuhkan.

3.3.6. Penyerahan Proyek

Setelah dilakukan perbaikan, kode sumber (*source code*) diserahkan pada pihak instansi guna proses perilisan. Perilisan dilakukan di bawah wewenang Dinas Kehutanan Provinsi Lampung serta Dinas Komunikasi dan Informatika (Diskominfotik), sehingga penelitian ini tidak mencangkup proses rilis. Pada tahap ini dilakukan *hands-off* repositori kode program, dokumen, dan *file* yang berkaitan dengan pengembangan aplikasi Sitanihut.

V. SIMPULAN DAN SARAN

5.1. Simpulan

Berdasarkan penelitian yang dilakukan, didapatkan kesimpulan yang dipaparkan sebagai berikut.

1. Pengembangan aplikasi *Offline* Android Sitanihut dengan pendekatan *Test Driven Development* (TDD) dilaksanakan dalam enam iterasi mencangkup 30 *user stories* dengan total 563 *unit test*.
2. Implementasi *Offline-First* Aplikasi Sitanihut telah dikembangkan menggunakan *local database* SQLite melalui *library* Room Persistence. Mekanisme sinkronisasi data dengan WorkManager memungkinkan sistem melakukan pengelolaan data lokal dan sinkronisasi dengan *server*. Penggunaan SyncStatus pada laporan mampu membedakan *item* mana yang sudah tersinkronisasi dan sesuai dengan *server* maupun yang mengalami perubahan di lokal.
3. Pengembangan aplikasi dengan metode *Test Driven Development* (TDD) menghasilkan kode program dengan *code coverage* tinggi. Dibuktikan dengan pengujian *code coverage* mencapai 95.53%. Mayoritas logika bisnis dan skenario penggunaan aplikasi telah ter*cover* oleh pengujian *unit*, sehingga mengurangi potensi *bug* regresi pada pengembangan selanjutnya.
4. Hasil pengujian *McCabe Cyclomatic Complexity* (MCC) memperoleh skor sebesar 2,36 dari ambang batas 10, skor *Cognitive Complexity* (CoCo) sebesar 1,77 dari ambang batas 15, rasio *technical debt* 0.4%, dan *maintainability rating* A dengan *tools* SonarQube. Hal ini menunjukkan bahwa alur logika program yang dikembangkan dengan penerapan TDD berada dalam batas kompleksitas dan *technical debt* yang rendah sehingga program lebih mudah untuk dipahami dan dimodifikasi.

5. Hasil pengujian *User Acceptance Test* (UAT) menunjukkan bahwa aplikasi Sitanihut dapat diterima dengan skor 100% berjalan baik.

5.2. Saran

Berdasarkan penelitian yang dilakukan, terdapat keterbatasan pada pengukuran *maintainability* menggunakan *Maintainability Index* (MI) karena tidak tersedianya penghitungan metrik *Halstead Volume* (HV) pada *tools* yang digunakan. Penelitian selanjutnya dapat mengeksplorasi lebih jauh *tools* yang dapat digunakan terkait metrik ini. Dapat juga dilakukan eksplorasi metrik pengukuran *maintainability* lain yang tidak bergantung pada HV. Selain itu, penerapan TDD pada pengembangan aplikasi ini dilakukan terbatas pada pengujian unit logika bisnis. Namun pengujian tersebut tidak menyentuh UI/antarmuka, sehingga pengujian navigasi dan alur bergantung pada *black box testing* manual. Penelitian selanjutnya dapat menerapkan pengujian otomatis tidak hanya pada *unit test* namun juga mencangkup *integration/*UI *test*, dalam hal ini dengan *framework* Espresso misalnya guna memastikan validitas alur dan interaksi UI tidak hanya logika bisnis yang ada.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

[1] Kementerian Kehutanan, ‘Visi Misi Kementerian Kehutanan’, Kementerian Kehutanan Republik Indonesia. Accessed: Jan. 22, 2026. [Online]. Available: https://www.kehutanan.go.id

[2] Dinas Kehutanan Provinsi Lampung, *Buku Saku Kehutanan tahun 2025*. Bandar Lampung: Dinas Kehutanan Provinsi Lampung, 2025.

[3] Badan Pusat Statistik Provinsi Lampung, ‘10,63% Rumah Tangga di Provinsi Lampung Memiliki Komputer/Laptop/Tablet - Berita dan Siaran Pers’, Badan Pusat Statistik Provinsi Lampung. Accessed: Nov. 12, 2025. [Online]. Available: https://lampung.bps.go.id/id/news/2024/04/01/294/10-63-persen-rumah-tangga-di-provinsi-lampung-memiliki-komputer-laptop-tablet.html

[4] Badan Pusat Statistik Indonesia, ‘Persentase Rumah Tangga yang Memiliki/Menguasai Telepon Seluler Menurut Provinsi dan Klasifikasi Daerah - Tabel Statistik’, Badan Pusat Statistik Indonesia. Accessed: Nov. 12, 2025. [Online]. Available: https://www.bps.go.id/id/statistics-table/2/MzkxIzI=/persentase-rumah-tangga-yang-memiliki-menguasai--telepon-seluler-menurut-provinsi-dan-klasifikasi-daerah.html

[5] Badan Pusat Statistik Provinsi Lampung, ‘Statistik Potensi Desa Provinsi Lampung 2024’. Accessed: Oct. 20, 2025. [Online]. Available: https://lampung.bps.go.id/id/publication/2024/12/30/b80dda48bfdbf74f322e4f47/statistik-potensi-desa-provinsi-lampung-2024.html

[6] Android Developers, ‘Save data in a local database using Room | App data and files’, Android Developers. Accessed: Aug. 27, 2025. [Online]. Available: https://developer.android.com/training/data-storage/room

[7] D. Janzen and H. Saiedian, ‘Does Test-Driven Development Really Improve Software Design Quality?’, *IEEE Softw.*, vol. 25, no. 2, pp. 77–84, Mar. 2008, doi: 10.1109/MS.2008.34.

[8] R. C. Martin, *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Prentice Hall Press, 2018.

[9] Kementrian LHK, ‘Rencana Kerja BP2SDM Tahun 2024’. Kementrian Lingkungan Hidup dan Kehutanan, 2024.

[10] Kementrian Kehutanan Republik Indonesia, ‘Kesatuan Pengelolaan Hutan (KPH)’, Kementrian Kehutanan. Accessed: Nov. 10, 2025. [Online]. Available: https://www.kehutanan.go.id/program/KESATUAN-PENGELOLAAN-HUTAN--

[11] L. Alwakeel, K. Lano, and H. Alfraihi, ‘AppCraft: Model-Driven Development Framework for Mobile Applications’, *IEEE Access*, vol. 13, pp. 23658–23699, 2025, doi: 10.1109/ACCESS.2025.3536321.

[12] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, and G. Ghinea, 'An empirical investigation of performance overhead in cross-platform mobile development frameworks', *Empir. Softw. Eng.*, vol. 25, no. 4, pp. 2997–3040, Jul. 2020, doi: 10.1007/s10664-020-09827-6.

[13] Android Developers, 'Data and file storage overview | App data and files', Android Developers. Accessed: Aug. 27, 2025. [Online]. Available: https://developer.android.com/training/data-storage

[14] Stat Counters, 'Mobile Operating System Market Share Indonesia', StatCounter Global Stats. Accessed: Aug. 26, 2025. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/indonesia

[15] Android Developers, 'Android's Kotlin-first approach', Android Developers. Accessed: Aug. 27, 2025. [Online]. Available: https://developer.android.com/kotlin/first

[16] R. Rua and J. Saraiva, 'A large-scale empirical study on mobile performance: energy, run-time and memory', *Empir. Softw. Eng.*, vol. 29, no. 1, pp. 1–56, Jan. 2024, doi: 10.1007/s10664-023-10391-y.

[17] Android Developers, 'Why Compose | Jetpack Compose', Android Developers. Accessed: Aug. 27, 2025. [Online]. Available: https://developer.android.com/develop/ui/compose/why-adopt

[18] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 7th edn. New York, NY: McGraw-Hill, 2010.

[19] G. E. Iyawa, 'Personal Extreme Programming: Exploring Developers' Adoption', in *AMCIS 2020 Proceedings*, virtual: Sheffield Hallam University Research Archive, Aug. 2020. [Online]. Available: https://aisel.aisnet.org/amcis2020/it_project_mgmt/it_project_mgmt/1

[20] Y. Dzhurov, I. Krasteva, and S. Ilieva, 'Personal Extreme Programming – An Agile Process for Autonomous Developers', in *Software, Services & Semantic Technologies*, Sofia, Bulgaria: Demetra EOOD, Oct. 2009. [Online]. Available: https://research.uni-sofia.bg/handle/10506/251

[21] K. Beck, *Test Driven Development: By Example*. Boston, MA: Addison-Wesley, 2002.

[22] S. Alsaqqa, S. Sawalha, and H. Abdel-Nabi, 'Agile Software Development: Methodologies and Trends', *Int. J. Interact. Mob. Technol. IJIM*, vol. 14, no. 11, pp. 246–270, Jul. 2020, doi: 10.3991/ijim.v14i11.13269.

[23] V. Monochristou and M. Vlachopoulou, 'Requirements Specification using User Stories', in *Agile Software Development Quality Assurance*, IGI Global Scientific Publishing, 2007, pp. 71–89. doi: 10.4018/978-1-59904-216-9.ch004.

[24] S. H. Pothineni, 'Offline-First Mobile Architecture: Enhancing Usability and Resilience in Mobile Systems', *J. Artif. Intell. Gen. Sci. JAIGS ISSN3006-4023*, vol. 7, no. 01, pp. 320–326, Dec. 2024, doi: 10.60087/jaigs.v7i01.387.

[25] Android Developers, 'Build an offline-first app | App architecture | Android Developers', Android Developers. Accessed: Aug. 27, 2025. [Online]. Available: https://developer.android.com/topic/architecture/data-layer/offline-first

[26] OMG Object Management Group, 'Unified Modeling Language, v2.5.1'. Object Management Group (OMG), Dec. 2017. Accessed: Sep. 02, 2026. [Online]. Available: https/www.omg.org/spec/UML/

[27] IEEE Computer Society, ‘Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) V4.0a’, IEEE Computer Society, Standard, May 2025.
[28] V. Khorikov, *Unit Testing Principles, Practices, and Patterns*. Simon and Schuster, 2020. [Online]. Available: https://books.google.co.id/books?id=rDszEAAAQBAJ
[29] I. Sommerville, *Software Engineering*, 10th edn. Courier Westford, USA: Pearson, 2016.
[30] ISO/IEC/IEEE, ‘ISO/IEC 25010:2023’, International Organization for Standardization, International Electrotechnical Commission, Switzerland, ISO/IEC 25010:2023, 2023. Accessed: Apr. 09, 2026. [Online]. Available: https://www.iso.org/standard/78176.html
[31] L. Ardito, R. Coppola, L. Barbato, and D. Verga, ‘A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review’, *Sci. Program.*, vol. 2020, no. 1, p. 8840389, 2020, doi: 10.1155/2020/8840389.
[32] T. Heričko and B. Šumak, ‘Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems’, *Appl. Sci.*, vol. 13, no. 5, p. 2972, Jan. 2023, doi: 10.3390/app13052972.
[33] Siti Rochimah, ‘A Maintainability Framework to Ensure the Software Quality in Object-Oriented Programming’, *IEEE Access*, vol. 13, 2025, doi: 10.1109/ACCESS.2025.3633265.
[34] T. J. McCabe, ‘A Complexity Measure’, *IEEE Trans. Softw. Eng.*, vol. 2, no. 04, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.
[35] G. A. Campbell, ‘{Cognitive Complexity} a new way of measuring understandability’, SonarSource S.A., Switzerland, White Paper Version 1.7, Apr. 2021. Accessed: Jan. 22, 2026. [Online]. Available: https://www.sonarsource.com/resources/cognitive-complexity/
[36] detekt contributors, ‘Rule: CognitiveComplexMethod’, detekt. Accessed: Jan. 22, 2026. [Online]. Available: https://detekt.dev/docs/rules/complexity/
[37] R. Cerquozzi *et al.*, ‘ISTQB Certified Tester - Foundation Level Syllabus v4.0.1’. International Software Testing Qualifications Board, 2024.
[38] P. Strečanský, S. Chren, and B. Rossi, ‘Comparing Maintainability Index, SIG Method, and SQALE for Technical Debt Identification’, *Sci. Program.*, vol. 2020, no. 1, p. 2976564, 2020, doi: 10.1155/2020/2976564.
[39] SonarSource, ‘Understanding measures and metrics | SonarQube Server | Sonar Documentation’. Accessed: Apr. 06, 2026. [Online]. Available: https://docs.sonarsource.com/sonarqube-server/user-guide/code-metrics/metrics-definition
[40] SonarSource, ‘Bugs and Vulnerabilities are 1st Class Citizens in SonarQube Quality Model along with Code Smells’. Accessed: Apr. 10, 2026. [Online]. Available: https://www.sonarsource.com/blog/bugs-and-vulnerabilities-are-1st-class-citizens-in-sonarqube-quality-model-along-with-code-smells
[41] ISO/IEC/IEEE, ‘ISO/IEC/IEEE 29119-1:2022 Software and systems engineering — Software testing — Part 1: General concepts’, International Organization for Standardization, International Electrotechnical Commission, dan Institute of Electrical and Electronics Engineers, Geneva, Switzerland, 29119–1:2022, 2022. Accessed: Feb. 10, 2026. [Online]. Available: https://www.iso.org/standard/81291.html

[42] G. I. Marthasari, W. Suharso, and F. Ardiansyah, ‘Personal Extreme Programming with MoSCoW Prioritization for Developing Library Information System’, *Int. Conf. Electr. Eng. Comput. Sci. Inform. EECSI*, 2018.

[43] A. Musofa, S. Fadli, W. Murniati, and H. Fahmi, ‘Implementasi Metode Personal Extreme Programming (PXP) Pada Pengembangan Aplikasi Buku Tamu(Studi Kasus : Dinas Komunikasi Dan Informatika Kabupaten Lombok Tengah)’, *J. Teknol. Inf. Dan Ilmu Komput.*, vol. 12, no. 5, pp. 985–996, Oct. 2025, doi: 10.25126/jtiik.2025125.

[44] W. Ren and S. Barrett, ‘Test-driven development, engagement in activity, and maintainability: An empirical study’, *IET Softw.*, vol. 17, no. 4, pp. 509–525, 2023, doi: 10.1049/sfw2.12135.

[45] B. Papis, K. Grochowski, K. Subzda, and K. Sijko, ‘Experimental Evaluation of Test-Driven Development With Interns Working on a Real Industrial Project’, *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1644–166, May 2022, doi: https://doi.org/10.1109/TSE.2020.302752.

[46] A. Räty, ‘Clean Architecture Android application for surveying HVAC systems’, Diplomityö, A. Räty, Finlandia, 2025. Accessed: Mar. 03, 2026. [Online]. Available: https://oulurepo.oulu.fi/handle/10024/59994

[47] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, and J. M. Patel, ‘SQLite: past, present, and future’, *Proc. VLDB Endow.*, vol. 15, no. 12, Aug. 2022, doi: 10.14778/3554821.3554842.