Loyola University Chicago

Department of Computer Science

Test Plan & State Machine Design Document

For Android Timer Application

COMP 313/413 – Intermediate Object-Oriented Design

Dr. Robert Yacobellis

Submitted By:

Halah, Ahmad
Kofira, Zach
Alkhalil, Mahran
Zulaik, Che Alyssa
Goldberg-Finkelstein, Jonah
Serek, Kassidy

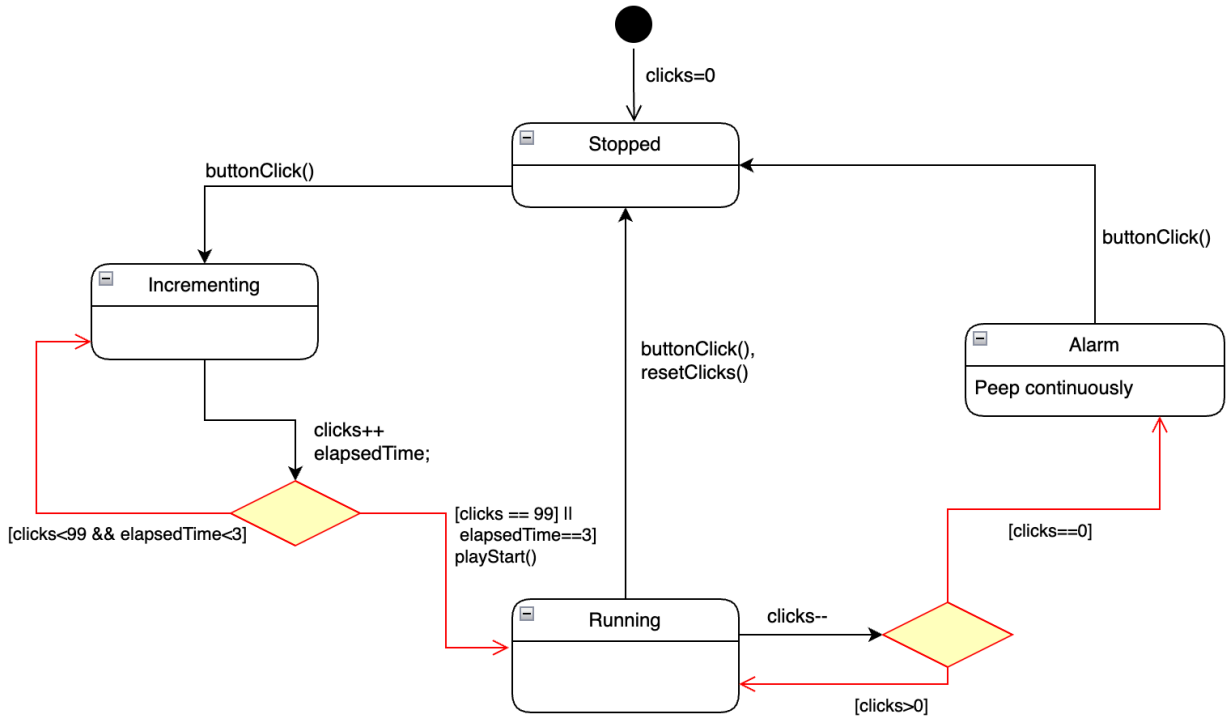
November 8, 2025

1. Overview

This document outlines the test plan and state machine design for an Android timer application. The project follows Test-Driven Development (TDD) methodology and implements the State Pattern for managing timer behavior.

2. State Machine Design

2.1 State Diagram



2.2 State Descriptions

State	Description	Entry Condition	Exit Condition
Stopped	Initial state, timer at zero	Application start, timer cancelled, alarm stopped	Button pressed
Incrementing	User setting timer value	Button pressed from Stopped	3-second timeout OR reaching 99
Running	Timer counting down	Auto-start after timeout or max	Button pressed OR reaching zero
Alarm	Continuous alarm beeping	Timer reaches zero	Button pressed

2.3 State Transitions

From	To	Event	Guard	Effect
Stopped	Incrementing	buttonPress	clicks < 99	clicks++
Incrementing	Incrementing	buttonPress	clicks < 99	clicks++, recordTime()
Incrementing	Running	timeout	timeElapsed >= 3 OR clicks == 99	playStartBeep()
Running	Running	timeElapsed(1sec)	clicks > 0	clicks--
Running	Stopped	buttonPress	-	stopAlarm(), clicks = 0
Running	Alarm	timeElapsed(1sec)	clicks == 0	startContinuousAlarm()
Alarm	Stopped	buttonPress	-	stopAlarm(), clicks = 0

3. Proposed Tests

This project follows a **TDD approach**, the following tests are the major tests will be implemented for the functional requirements, while additional tests will be added upon implementation:

- State Machine core tests
- Model view and Integration tests
- Activity User Interface tests

3.1 State Machine core tests:

- testButtonPressTransitionsToIncrementingState

What it tests: Initial requirement - pressing button when stopped increments time
Needed for: This is the entry point to the timer. If this doesn't work, nothing else works.
Requirement: If the button is pressed when the timer is stopped, the time is incremented by one

- testThreeSecondTimeoutTransitionsToRunning

What it tests: Auto-start after 3-second delay
Needed for: Tests the time-based guard with injected Time provider
Requirement: If three seconds elapse from the most recent time the button was pressed, then the timer beeps once and starts running
Role: Test dependency injection working, guard logic, no static time dependency

- testIncrementingStopsAt99

What it tests: Maximum value constraint
Needed for: Tests boundary condition and transition to Running at max
Requirement: If the time reaches the preset maximum of 99 the timer acts the same way as if three seconds had elapsed
Role: Tests guard logic ([clicks < 99])

- testTimeElapsedDecrementsTime

What it tests: Countdown behavior
Needed for: Core timer functionality - counting down
Requirement: While running, the timer subtracts one from the time for every second that elapses
Role: Tests running state behavior, self-transition with guard

- testTimeReachingZeroTransitionsToAlarm

What it tests: Transition to alarm when countdown finishes
Needed for: Tests automatic state transition without user input
Requirement: If the timer is running and the time reaches zero by itself, then the timer stops and the alarm starts beeping
Role: Tests guard checking ([clicks == 0]), state transition logic

- testButtonPressDuringRunningTransitionsToStopped

What it tests: Cancel functionality
Needed for: Tests button's multi-function behavior - acts as cancel during running
Requirement: If the timer is running and the button is pressed, the timer stops and the time is reset to zero
Role: Tests the same button, different behavior based on state

- testButtonPressInAlarmTransitionsToStopped

What it tests: Stopping alarm with button
Needed for: Completes the state cycle
Requirement: If the alarm is sounding and the button is pressed, the alarm stops sounding
Role: Tests how is the button behavior depends on current state

- testCompleteTimerFlowFromStartToAlarm

What it tests: Complete through all states
Needed for: Integration test showing all states work together
Flow: Stopped → Incrementing → Running → Alarm → Stopped
Role: Tests the state machine correctness, all transitions work in sequence

3.2 Model view and Integration tests:

- testTimerAutoStartsAfterThreeSeconds : tests if the view model coordinates timer with state machine.
- testButtonPressUpdatesUiState : tests if the view model translates state to UI state (remaining time).
- testDisplayChangesOneSecondAfterAutoStart: test after 3-sec timeout, display doesn't change until next click

3.3 Activity User Interface:

- testActivityPlaysBeepOnStartEffect: tests if the activity responds to play start beep effect
- testConfChangePreservesTimerState: tests if the activity preserves configurations changes like rotation.
- testDisplayFormatsTwoDigits : tests if display shows two digits (00...99)