# Regression Testing Plan

- Dynamic Slice Technique Example BT8–

Dependable Software Laboratory
Sun Hwi Lee

# Contents

- Dynamic Slice Technique
- Dynamic Program Slice for Test Cases
- Reference

# The Dynamic Slice Technique

- During the off-line processing, find the dynamic program slices with respect to the program output for all test cases in the regression test suite. Then, after the program is modified, rerun the new program on only those test cases whose dynamic program slices contain a modified statement.

# The Dynamic Slice Technique

- Test Case의 결과와 관련된 부분에서 수정된 부분이 존재하면 테스팅

| Testcase | Input | | | Output | |
|---|---|---|---|---|---|
| | a | b | c | class | area |
| $T_1$ | 2 | 2 | 2 | equilateral | 1.73 |
| $T_2$ | 4 | 4 | 3 | isosceles | 5.56 |
| $T_3$ | 5 | 4 | 3 | right | 6.00 |
| $T_4$ | 6 | 5 | 4 | scalene | 9.92 |
| $T_5$ | 3 | 3 | 3 | equilateral | 2.60 |
| $T_6$ | 4 | 3 | 3 | scalene | 4.47 |

*failures!*

Example Test Suite

```
S1:  read (a, b, c);
S2:  class := scalene;
S3:  if a = b or b = a
S4:     class := isosceles;
S5:  if a*a = b*b + c*c
S6:     class := right;
S7:  if a = b and b = c
S8:     class := equilateral;
S9:  case class of
S10:    right       : area := b*c / 2;
S11:    equilateral : area :=a*a * sqrt(3)/4;
S12:    otherwise   : s := (a+b+c)/2;
S13:                  area := sqrt(s*(s-a)*(s-b)*(s-c));
     end;
S14: write(class, area);
```

*the changed statement is outside the dynamic slice!*

Dynamic Program Slice for T1

```
S1:  read (a, b, c);
S2:  class := scalene;
S3:  if a = b or b = a
S4:     class := isosceles;
S5:  if a*a = b*b + c*c
S6:     class := right;
S7:  if a = b and b = c
S8:     class := equilateral;
S9:  case class of
S10:    right       : area := b*c / 2;
S11:    equilateral : area :=a*a * sqrt(3)/4;
S12:    otherwise   : s := (a+b+c)/2;
S13:                  area := sqrt(s*(s-a)*(s-b)*(s-c));
     end;
S14: write(class, area);
```

*the changed statement is outside the dynamic slice!*

Dynamic Program Slice for T4

# The Dynamic Slice Technique

## Process

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.000 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Time Display |
| **Input** | btn = 'c';<br>*stateData.cs = TimeDisplay;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == StopwatchDisplay;<br>Flag_StopwatchDisplay == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void stopwatchDisplay(TimeData *timeData) {
    stopwatchDisplayInterface(timeData);

    Flag_StopwatchDisplay = 1;
}
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:       timeDisplay(timeData);                  break;
    case TimeSetting:       timeSetting(timeData);                  break;
    case AddVariable:       addVariable(stateData, timeData);       break;
    case MoveVariable:      moveVariable(stateData, timeData);      break;
    case MeasuringTimeOn:   measuringTimeOn(stateData, timeData);   break;
    case MeasuringTimeOff:  measuringTimeOff(stateData, timeData);  break;
    case StopwatchDisplay:  stopwatchDisplay(timeData);             break;
    case LapTime:           laptime(stateData, timeData);           break;
    case MeasuringTimeReset: measuringTimeReset(stateData, timeData); break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
} else if (buttonCCommand) {
    if ((*stateData).cs == TimeDisplay) {
        (*stateData).cs = StopwatchDisplay;
    } else if ((*stateData).cs == TimeSetting) {
        (*stateData).cs = MoveVariable;
    } else if ((*stateData).cs == StopwatchDisplay) {
        (*stateData).cs = TimeDisplay;
    }
} else if (buttonDCommand) {
    (*stateData).bs = BacklightOn;
}
```

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.001 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Time Display |
| **Input** | btn = 'a';<br>*stateData.cs = TimeDisplay;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == TimeSetting;<br>Flag_TimeSetting == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void timeSetting(TimeData *timeData) {
    timeSettingInterface(timeData);

    Flag_TimeSetting = 1;
}
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:          timeDisplay(timeData);                  break;
    case TimeSetting:          timeSetting(timeData);                  break;
    case AddVariable:          addVariable(stateData, timeData);       break;
    case MoveVariable:         moveVariable(stateData, timeData);      break;
    case MeasuringTimeOn:      measuringTimeOn(stateData, timeData);   break;
    case MeasuringTimeOff:     measuringTimeOff(stateData, timeData);  break;
    case StopwatchDisplay:     stopwatchDisplay(timeData);             break;
    case LapTime:              laptime(stateData, timeData);           break;
    case MeasuringTimeReset:   measuringTimeReset(stateData, timeData); break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
if (buttonACommand) {
    if ((*stateData).cs == TimeDisplay) {
        (*stateData).cs = TimeSetting;
    } else if ((*stateData).cs == TimeSetting) {
        (*stateData).cs = TimeDisplay;
    } else if ((*stateData).cs == StopwatchDisplay) {
        if ((*stateData).ss == MeasuringTimeOff) {
            (*stateData).cs = MeasuringTimeReset;
        } else if ((*stateData).ss == MeasuringTimeOn
                || (*stateData).ps == LapTime) {
            (*stateData).cs = LapTime;
        }
    }
```

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.002 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Time Setting |
| **Input** | btn = 'c';<br>*stateData.cs = TimeSetting;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == TimeSetting;<br>Flag_MoveVariable == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void moveVariable(StateData *stateData, TimeData *timeData)
    (*timeData).cursor++;
    if ((*timeData).cursor > CDate) {
        (*timeData).cursor = CSec;
    }

    (*stateData).cs = TimeSetting;

    Flag_MoveVariable = 1;
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:          timeDisplay(timeData);                  break;
    case TimeSetting:          timeSetting(timeData);                  break;
    case AddVariable:          addVariable(stateData, timeData);       break;
    case MoveVariable:         moveVariable(stateData, timeData);      break;
    case MeasuringTimeOn:      measuringTimeOn(stateData, timeData);   break;
    case MeasuringTimeOff:     measuringTimeOff(stateData, timeData);  break;
    case StopwatchDisplay:     stopwatchDisplay(timeData);             break;
    case LapTime:              laptime(stateData, timeData);           break;
    case MeasuringTimeReset:   measuringTimeReset(stateData, timeData); break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
} else if (buttonCCommand) {
    if ((*stateData).cs == TimeDisplay) {
        (*stateData).cs = StopwatchDisplay;
    } else if ((*stateData).cs == TimeSetting) {
        (*stateData).cs = MoveVariable;
    } else if ((*stateData).cs == StopwatchDisplay) {
        (*stateData).cs = TimeDisplay;
    }
} else if (buttonDCommand) {
    (*stateData).bs = BacklightOn;
}
```

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.003 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Time Setting |
| **Input** | btn = 'b';<br>*stateData.cs = TimeSetting;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == TimeSetting;<br>Flag_addVariable == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void addVariable(StateData *stateData, TimeData *timeData) {
    switch ((*timeData).cursor) {
    case CSec:
        (*timeData).sec++;
```

```
default: break;
}
(*stateData).cs = TimeSetting;

Flag_addVariable = 1;
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:          timeDisplay(timeData);                  break;
    case TimeSetting:          timeSetting(timeData);                  break;
    case AddVariable:          addVariable(stateData, timeData);       break;
    case MoveVariable:         moveVariable(stateData, timeData);      break;
    case MeasuringTimeOn:      measuringTimeOn(stateData, timeData);   break;
    case MeasuringTimeOff:     measuringTimeOff(stateData, timeData);  break;
    case StopwatchDisplay:     stopwatchDisplay(timeData);             break;
    case LapTime:              laptime(stateData, timeData);           break;
    case MeasuringTimeReset:   measuringTimeReset(stateData, timeData); break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
else if (buttonBCommand) {
    if ((*stateData).cs == StopwatchDisplay) {
        if ((*stateData).ss == MeasuringTimeOn) {
            if ((*stateData).ps == LapTime) {
                (*stateData).cs = MeasuringTimeOn;
            } else {
                (*stateData).cs = MeasuringTimeOff;
            }
        } else if ((*stateData).ss == MeasuringTimeOff
                || (*stateData).ps == LapTime) {
            (*stateData).cs = MeasuringTimeOn;
        }
    } else if ((*stateData).cs == TimeSetting) {
        (*stateData).cs = AddVariable;
    }
```

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.004 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Time Setting |
| **Input** | btn = 'a';<br>*stateData.cs = TimeSetting;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == TimeDisplay;<br>Flag_TimeDisplay == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void timeDisplay(TimeData *timeData) {
    timeDisplayInterface(timeData);

    Flag_TimeDisplay = 1;
}
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:           timeDisplay(timeData);                  break;
    case TimeSetting:           timeSetting(timeData);                  break;
    case AddVariable:           addVariable(stateData, timeData);       break;
    case MoveVariable:          moveVariable(stateData, timeData);      break;
    case MeasuringTimeOn:       measuringTimeOn(stateData, timeData);   break;
    case MeasuringTimeOff:      measuringTimeOff(stateData, timeData);  break;
    case StopwatchDisplay:      stopwatchDisplay(timeData);             break;
    case LapTime:               laptime(stateData, timeData);           break;
    case MeasuringTimeReset:    measuringTimeReset(stateData, timeData);break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
if (buttonACommand) {
    if ((*stateData).cs == TimeDisplay) {
        (*stateData).cs = TimeSetting;
    } else if ((*stateData).cs == TimeSetting) {
        (*stateData).cs = TimeDisplay;
    } else if ((*stateData).cs == StopwatchDisplay) {
        if ((*stateData).ss == MeasuringTimeOff) {
            (*stateData).cs = MeasuringTimeReset;
        } else if ((*stateData).ss == MeasuringTimeOn
                || (*stateData).ps == LapTime) {
            (*stateData).cs = LapTime;
        }
    }
```

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.005 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Stopwatch Display |
| **Input** | btn = 'c';<br>*stateData.cs = StopwatchDisplay;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == TimeDiplay;<br>Flag_TimeDisplay == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void timeDisplay(TimeData *timeData) {
    timeDisplayInterface(timeData);

    Flag_TimeDisplay = 1;
}
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:       timeDisplay(timeData);                  break;
    case TimeSetting:       timeSetting(timeData);                  break;
    case AddVariable:       addVariable(stateData, timeData);       break;
    case MoveVariable:      moveVariable(stateData, timeData);      break;
    case MeasuringTimeOn:   measuringTimeOn(stateData, timeData);   break;
    case MeasuringTimeOff:  measuringTimeOff(stateData, timeData);  break;
    case StopwatchDisplay:  stopwatchDisplay(timeData);             break;
    case LapTime:           laptime(stateData, timeData);           break;
    case MeasuringTimeReset: measuringTimeReset(stateData, timeData); break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
    } else if (buttonCCommand) {
        if ((*stateData).cs == TimeDisplay) {
            (*stateData).cs = StopwatchDisplay;
        } else if ((*stateData).cs == TimeSetting) {
            (*stateData).cs = MoveVariable;
        } else if ((*stateData).cs == StopwatchDisplay) {
            (*stateData).cs = TimeDisplay;
        }
```

# Dynamic Program Slice for Test Cases

| Identifier | UT.000.006 |
|---|---|
| **Controller** | 2.1.1 Time Controller |
| **State** | Stopwatch Display |
| **Input** | btn = 'b';<br>*stateData.cs = StopwatchDisplay;<br>*stateData.ss = MeasuringTimeOn;<br>*stateData.ps != LapTime;<br>changeState(*stateData);<br>timeController(*stateData, *timeData); |
| **Expected Output** | *stateData.cs == StopwatchDisplay;<br>Flag_measuringTimeOff == 1; |

```
typedef struct {
    State cs; // Current State
    State ps; // Previous State
    State ss; // Stopwatch State
    State bs; // Backlight State
} StateData;
```

```
void measuringTimeOff(StateData *stateData, TimeData *timeData) {
    (*stateData).cs = StopwatchDisplay;
    (*stateData).ss = MeasuringTimeOff;

    (*timeData).stStatus = false;

    Flag_measuringTimeOff = 1;
}
```

Main.c | Controller_TimeController.c

```
 * Controller_TimeController.c
#include "Header_Controller.h"
#include "Header_DataProcess.h"

void timeController(StateData *stateData, TimeData *timeData) {
    switch ((*stateData).cs) {
    case TimeDisplay:          timeDisplay(timeData);                break;
    case TimeSetting:          timeSetting(timeData);                break;
    case AddVariable:          addVariable(stateData, timeData);     break;
    case MoveVariable:         moveVariable(stateData, timeData);    break;
    case MeasuringTimeOn:      measuringTimeOn(stateData, timeData); break;
    case MeasuringTimeOff:     measuringTimeOff(stateData, timeData);break;
    case StopwatchDisplay:     stopwatchDisplay(timeData);           break;
    case LapTime:              laptime(stateData, timeData);         break;
    case MeasuringTimeReset:   measuringTimeReset(stateData, timeData); break;
    default: break;
    }
}
```

# Dynamic Program Slice for Test Cases

```
void changeState(StateData *stateData) {
    bool buttonACommand = false;
    bool buttonBCommand = false;
    bool buttonCCommand = false;
    bool buttonDCommand = false;
    char btn = '\0';

    // Getting Button Commands
    if(_kbhit()){
        btn = getch();
        fflush(stdin);
    }

    switch (btn) {
    case 'a': case 'A': buttonACommand = buttonAInterface(); break;
    case 'b': case 'B': buttonBCommand = buttonBInterface(); break;
    case 'c': case 'C': buttonCCommand = buttonCInterface(); break;
    case 'd': case 'D': buttonDCommand = buttonDInterface(); break;
    }

    // Set Priority
    if ((buttonACommand && buttonBCommand) || (buttonACommand && buttonCCommand)
            || (buttonACommand && buttonDCommand)) {
        buttonACommand = false;
    }
    if ((buttonBCommand && buttonCCommand)
            || (buttonBCommand && buttonDCommand)) {
        buttonBCommand = false;
    }
    if ((buttonCCommand && buttonDCommand)) {
        buttonCCommand = false;
    }
```

```
} else if (buttonBCommand) {
    if ((*stateData).cs == StopwatchDisplay) {
        if ((*stateData).ss == MeasuringTimeOn) {
            if ((*stateData).ps == LapTime) {
                (*stateData).cs = MeasuringTimeOn;
            } else {
                (*stateData).cs = MeasuringTimeOff;
            }
        } else if ((*stateData).ss == MeasuringTimeOff
                || (*stateData).ps == LapTime) {
            (*stateData).cs = MeasuringTimeOn;
        }
    } else if ((*stateData).cs == TimeSetting) {
        (*stateData).cs = AddVariable;
    }
```

# Reference


- Incremental regression testing, Agrawal, H. and Horgan, J.R. and Krauser, E.W. and London, S.A., Software Maintenance, 1993. CSM-93, Proceedings., Conference on, 348-357, 1993, IEEE