สัปดาห์ที่ 1: Introduction to Software Testing

หัวข้อ 1.1: ความหมายและความสำคัญของ Software Testing

คำถามเปิดเรื่อง:

มีใครเคยใช้แอพหรือเว็บไซต์แล้วเจอ bug บ้าง เช่น กดปุ่มแล้วไม่ทำงาน หรือข้อมูลหาย

ความหมายของ Software Testing:

Software Testing คือกระบวนการตรวจสอบและประเมินระบบซอฟต์แวร์เพื่อ:

- ยืนยันว่าซอฟต์แวร์ทำงานตาม requirements ที่กำหนด
- ค้นหาข้อบกพร่อง (defects/bugs) ก่อนส่งมอบให้ผู้ใช้
- ประเมินคุณภาพของซอฟต์แวร์
- สร้างความมั่นใจว่าซอฟต์แวร์พร้อมใช้งาน

ทำไมต้อง Test?

สถิติที่น่าสนใจ:

- 🔴 88% ของ mobile apps มี bugs ที่ส่งผลต่อ user experience
- 🔴 60% ของผู้ใช้ จะลบแอพทันทีหากเจอ bugs ใน session แรก
- 🔴 Software bugs ทำให้เศรษฐกิจสหรัฐฯ เสียหายมากกว่า $2 trillion ต่อปี

ตัวอย่างจากชีวิตจริง:

1. Knight Capital (2012)
    - เสียเงิน $440 million ในเวลา 45 นาที
    - สาเหตุ: Bug ในระบบ trading algorithm
    - บริษัทล้มละลายภายใน 1 สัปดาห์
2. Therac-25 Medical Equipment (1985-1987)
    - เครื่องฉายรังสีรักษามะเร็ง
    - Race condition bug ทำให้ฉายรังสีเกินขนาด
    - ผู้เสียชีวิต 3 คน, บาดเจ็บสาหัส 3 คน
3. Ariane 5 Rocket (1996)
    - จรวดยุโรปมูลค่า $370 million
    - Integer overflow bug
    - ระเบิดหลังปล่อยเพียง 40 วินาที

Testing ≠ Quality Assurance

Aspect	Testing	Quality Assurance
Focus	หา bugs	ป้องกันไม่ให้เกิด bugs
Approach	Reactive	Proactive
When	หลัง development	ตลอด SDLC
Activities	Execute tests, find defects	Process improvement, reviews
Goal	Product verification	Process prevention

หัวข้อ 1.2: SDLC vs STLC

Software Development Life Cycle (SDLC)

Waterfall Model:
Requirements → Design → Implementation → Testing → Deployment → Maintenance

Agile/Scrum Model:
Sprint Planning → Development → Testing → Review → Deployment
(Repeat every 2-4 weeks)

Software Testing Life Cycle (STLC)

STLC เป็น subset ของ SDLC ที่เน้นเฉพาะกิจกรรมการทดสอบ:

6 Phases of STLC:

1. Requirement Analysis
    - ศึกษา requirements
    - ระบุ testable requirements
    - ทำ Requirement Traceability Matrix (RTM)
2. Test Planning
    - สร้าง Test Strategy
    - ประมาณการ effort และ resources
    - กำหนด test schedule
3. Test Case Development
    - เขียน test cases
    - เตรียม test data
    - Setup test environment
4. Test Environment Setup
    - ติดตั้ง hardware/software
    - Configure test data
    - Network setup
5. Test Execution
    - รัน test cases
    - บันทึกผลการทดสอบ
    - Report bugs
6. Test Closure
    - ประเมินผล test coverage
    - สร้าง test summary report
    - Lessons learned

Entry & Exit Criteria

Entry Criteria (**ก่อนเริ่ม** testing):

- Test plan approved
- Test cases reviewed
- Test environment ready
- Test data prepared
- Build deployed to test environment

Exit Criteria (สิ้นสุด testing phase):

- 100% test cases executed
- 90%+ test cases passed
- All critical bugs fixed
- Test summary report completed
- Sign-off from stakeholders

หัวข้อ 1.3: Cost of Quality & Cost of Bugs

Cost of Quality (CoQ)

Cost of Quality แบ่งเป็น 2 ประเภท:

1. Cost of Good Quality (Conformance):

- Prevention Costs: ค่าใช้จ่ายในการป้องกัน bugs
  - Training
  - Process improvement
  - Code reviews
  - Requirements analysis
- Appraisal Costs: ค่าใช้จ่ายในการหา bugs
  - Testing activities
  - Test automation
  - Quality audits

2. Cost of Poor Quality (Non-conformance):

- Internal Failure Costs: พบ bugs ก่อนส่งมอบ
  - Rework
  - Debugging time
  - Re-testing
- External Failure Costs: พบ bugs หลังส่งมอบ
  - Customer support
  - Warranty claims
  - Loss of reputation
  - Legal costs

Cost Amplification (1-10-100-1000 Rule)

Bug **ที่พบช้า** = **แก้ไขแพงขึ้นแบบ** exponential:

Phase	Relative Cost	Example Cost
Requirements	1x	$100
Design	10x	$1,000
Implementation	100x	$10,000
Testing	1,000x	$100,000
Production	10,000x+	$1,000,000+

ตัวอย่าง: Cost Analysis **จริง**

กรณีศึกษา: E-commerce Checkout Bug

Scenario: ระบบคำนวณส่วนลดผิด (ลด 100% แทนที่จะเป็น 10%)

ถ้าพบในแต่ละ phase:

1. Requirements Phase (Cost: $500)
   - นักวิเคราะห์เห็นว่า requirement คลุมเครือ

- แก้ไขโดยชี้แจง requirement ใหม่
- ใช้เวลา 4 ชม. × $125/hr = $500

2. Implementation Phase (Cost: $5,000)
   - Programmer เขียน code ผิด
   - QA พบตอน unit testing
   - ต้อง rewrite code + test ใหม่
   - ใช้เวลา 40 ชม. × $125/hr = $5,000
3. Production Phase (Cost: $500,000+)
   - ลูกค้า 1,000 คนสั่งซื้อฟรี (ควรจ่าย $500/คน)
   - เสียรายได้: $500,000
   - Emergency fix: $10,000
   - PR damage: ประเมินไม่ได้
   - Total: $510,000+

ROI of Testing
Investment in Testing:

- Salary QA Engineer: $50,000/year
- Testing tools: $10,000/year
- Total: $60,000/year

Potential Savings:

- Prevent 1 production bug like above: $500,000
- ROI: 733%

หัวข้อ 1.4: Testing vs Debugging vs QA (15 **นาที**)
สามสิ่งที่แตกต่างกัน:

SOFTWARE QUALITY ACTIVITIES		
TESTING	**DEBUGGING**	**QUALITY ASSURANCE**
Find bugs Execute tests Report defects Verification Done by Testers	Fix bugs Locate root cause Code analysis Done by Developers	Prevent bugs Process improvement Standards & reviews Validation Done by QA team & everyone

ตัวอย่างเปรียบเทียบ:

Scenario: ระบบ Login ไม่ทำงาน

Testing:

- ✅ Execute test case: "Login with valid credentials"
- ✅ Result: FAIL - Cannot login
- ✅ Report: "Bug #123 - Login button not working"
- ✅ Severity: High, Priority: High

Debugging:

- 🔍 Developer รับ bug report
- 🔍 Reproduce the issue
- 🔍 Debug code → พบว่า onClick event ไม่ถูก bind
- 🔍 Fix: Add event listener ให้ถูกต้อง
- 🔍 Verify fix works

Quality Assurance:

- 📋 Review coding standards
- 📋 Implement code review process
- 📋 Add checklist: "Verify all buttons have event handlers"
- 📋 Training: "Common JavaScript pitfalls"
- 📋 Goal: ป้องกันไม่ให้เกิด bug แบบนี้อีก

Verification vs Validation
Verification: "Are we building the product right?"

- ✅ ตรวจสอบว่าทำตาม specs หรือไม่
- ✅ Static testing (reviews, inspections)
- ✅ ตอบคำถาม: "Did we implement this correctly?"

Validation: "Are we building the right product?"

- ✅ ตรวจสอบว่าตรงความต้องการผู้ใช้หรือไม่
- ✅ Dynamic testing (functional testing, UAT)
- ✅ ตอบคำถาม: "Is this what users actually need?"

หัวข้อ 1.5: Software Defects และผลกระทบ

Software Defect **คืออะไร?**

Defect (Bug) = ความแตกต่างระหว่าง:

- Expected behavior (สิ่งที่ควรเป็น)
- Actual behavior (สิ่งที่เป็นจริง)

ประเภทของ Defects:

1. Functional Defects

- Feature ไม่ทำงานตาม requirements
- ตัวอย่าง: กดปุ่ม Submit แล้วไม่บันทึกข้อมูล

2. Performance Defects

- ระบบช้า, timeout
- ตัวอย่าง: หน้าเว็บโหลดนานเกิน 10 วินาที

3. Usability Defects

- ใช้งานยาก, สับสน
- ตัวอย่าง: ปุ่ม "Save" กับ "Cancel" อยู่ผิดตำแหน่ง

4. Security Defects

- ช่องโหว่ด้านความปลอดภัย
- ตัวอย่าง: SQL Injection, XSS

5. Compatibility Defects

- ไม่ทำงานบน browser/OS บางตัว
- ตัวอย่าง: แสดงผลผิดบน Firefox

Defect Attributes:
Severity (**ระดับความรุนแรง**):

- Critical: ระบบล่มทั้งหมด, data loss, security breach
- High: Feature หลักไม่ทำงาน, ไม่มี workaround
- Medium: Feature รองไม่ทำงาน, มี workaround ได้
- Low: ปัญหาเล็กน้อย, cosmetic issues

Priority (**ความสำคัญในการแก้ไข**):

- P0 (Immediate): แก้ทันที, block release
- P1 (High): แก้ใน sprint นี้
- P2 (Medium): แก้ใน sprint หน้า
- P3 (Low): แก้เมื่อมีเวลา

ตัวอย่างการจัดลำดับความสำคัญ:

Bug	Severity	Priority	Reasoning
Login ไม่ได้	Critical	P0	ไม่สามารถใช้งานระบบได้เลย
Search ช้ามาก	High	P1	Feature หลัก แต่ยังใช้ได้
Logo resolution ต่ำ	Low	P3	ไม่กระทบการใช้งาน
Help text มีตัวสะกดผิด	Low	P2	แก้ง่าย แต่กระทบ image

Defect Life Cycle:

New → Assigned → Open → Fixed → Retest → Verified → Closed
↓
Rejected (Not a bug)
↓
Deferred (แก้ใน version หน้า)
↓
Duplicate (ซ้ำกับ bug อื่น)

หัวข้อ 1.6: Demo - Library Management System - แนะนำระบบที่จะใช้ตลอดเทอม

Library Management
Features **หลัก**:

1. User Management
    - Register/Login/Logout
    - User profiles
    - Role-based access (Member, Librarian, Admin)
2. Book Management
    - Search books (by title, author, ISBN)
    - View book details
    - Add/Edit/Delete books (Librarian only)
3. Borrowing System
    - Borrow books
    - Return books
    - Reserve books
    - View borrowing history
4. Catalog Features
    - Categories and tags
    - Book ratings and reviews
    - Advanced search filters

Tech Stack:

- Frontend: HTML, Bootstrap, JavaScript
- Backend: Node.js, Express
- Database: MySQL
- Deployment: Docker

Demo Walkthrough:
1. User Registration & Login

- สร้าง account ใหม่
- Login เข้าระบบ
- ดู user profile

2. Book Search & Browse

- ค้นหาหนังสือ "JavaScript"
- ดู book details
- เรียงลำดับตาม rating

3. Borrow & Return Flow

- ยืมหนังสือ

- ดู "My Books"
- คืนหนังสือ

4. **แสดง** Bugs **ที่ฝังไว้** (5 นาที)

- 🐛 Bug #1: Search ด้วย special characters ทำให้ crash
- 🐛 Bug #2: ยืมได้เกิน limit
- 🐛 Bug #3: Password requirements ไม่ชัดเจน
- 🐛 Bug #4: Date validation ไม่ทำงาน

5. **แนะนำ** Bug Categories (3 นาที)

- Security bugs
- Business logic bugs
- Data validation bugs
- UI/UX bugs
- Performance bugs

6. **ภาพรวมงานตลอดเทอม**

Week 1: Find bugs (manual exploration)
Week 2-5: Write test cases
Week 6: Code review & static analysis
Week 7-8: Unit & integration tests
Week 9: System & E2E tests
Week 10-11: Automation
Week 12-13: Performance & security
Week 14: CI/CD & final integration

O]O

ส่วนที่ 2: Activities

Activity 1: Setup Development Environment & Bug Hunting

วัตถุประสงค์:

1. ติดตั้งเครื่องมือพื้นฐานที่ใช้ตลอดเทอม
2. เรียนรู้การ explore ระบบเพื่อหา bugs
3. เข้าใจการจัดหมวดหมู่ bugs
4. เขียน bug report ตามมาตรฐาน

Part 1: Environment Setup
1.1 Install Node.js
Windows:
Download from https://nodejs.org/
เลือก LTS version (v18.x หรือ v20.x)
Double-click installer และติดตั้งตามขั้นตอน

Verify installation
node --version
npm --version

macOS:
Using Homebrew
brew install node@18

Verify
node --version
npm --version

Linux (Ubuntu/Debian):
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

Verify
node --version
npm --version

1.2 Install Visual Studio Code

1. Download from: https://code.visualstudio.com/
2. Install extensions:
    - ESLint
    - Prettier
    - JavaScript (ES6) code snippets
    - Live Server (for testing HTML)

1.3 Install Git
Windows:
Download from https://git-scm.com/
Run installer

Configure

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

macOS/Linux:
# macOS
brew install git

# Linux
sudo apt-get install git

# Configure
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

1.4 Clone Library System Repository

```
# สร้าง folder สำหรับเก็บ project
mkdir ~/software-testing
cd ~/software-testing

# Clone repository (อาจารย์จะให้ URL จริง)
git clone <repository-url> library-system
cd library-system

# Install dependencies
npm install

# Start the application
npm start

# เปิด browser ไปที่ http://localhost:3000
```

1.5 Browser Developer Tools

เปิด Chrome DevTools:

- Windows/Linux: F12 หรือ Ctrl + Shift + I
- macOS: Cmd + Option + I

Tab **ที่สำคัญ**:

- Elements: ดู HTML/CSS
- Console: ดู JavaScript errors
- Network: ดู API requests
- Application: ดู cookies, localStorage

Activity 2: Bug Hunting Session

2.1 Exploratory Testing (15 นาที)

Mission: หา bugs ในระบบ Library Management

Test Scenarios:

Scenario 1: User Registration

Test Steps:

1. ไปที่หน้า Register
2. ลองกรอกข้อมูลต่างๆ:
   - Username สั้นเกินไป (< 3 ตัวอักษร)
   - Email format ผิด (test@, @test.com, test.com)
   - Password ง่ายเกินไป (123, abc)
   - Password ไม่ตรงกัน
3. ลองกด Submit หลายครั้งติดกัน
4. ลองใส่ special characters (@#$%^&*)

Scenario 2: Login

Test Steps:

1. Login ด้วย username/password ที่ไม่มีในระบบ
2. Login ด้วย password ผิด
3. ทิ้ง field ว่าง
4. Copy-paste password (ดูว่ามี whitespace หรือไม่)
5. ลอง SQL injection: admin' OR '1'='1

Scenario 3: Search Books

Test Steps:

1. Search ด้วย keyword ปกติ
2. Search ด้วย empty string
3. Search ด้วย special characters: <script>, ', ", %
4. Search ด้วย SQL injection attempts
5. Search ด้วย keyword ยาวมากๆ (1000+ characters)
6. Search ด้วยภาษาไทย, emoji, unicode

Scenario 4: Borrow Books

Test Steps:

1. ยืมหนังสือปกติ
2. ลองยืมหนังสือเดิมซ้ำ
3. ลองยืมเกิน limit (ถ้ามี)
4. ยืมโดยไม่ login
5. ยืมหนังสือที่ถูกยืมไปแล้ว
6. เปลี่ยน borrow date เป็นอดีต/อนาคตไกลมาก

Scenario 5: UI/UX Issues
Check:
1. ปุ่มและ links ทั้งหมดทำงานไหม
2. Error messages ชัดเจนหรือไม่
3. Form validation ทำงานตอนไหน (client/server)
4. Responsive design (ลอง resize browser)
5. Navigation สะดวกหรือไม่

2.2 Bug Classification (10 นาที)
เมื่อเจอ bug แล้ว ให้จัดหมวดหมู่:
A. Bug Type:

- 🔒 Security (SQL injection, XSS, authentication bypass)
- 💼 Business Logic (ยืมเกิน limit, คำนวณผิด)
- ✅ Data Validation (invalid input accepted)
- 💾 Database (data inconsistency, integrity)
- 🎨 UI/UX (layout issues, poor usability)
- ⚡ Performance (slow response, memory leak)

B. Severity:

- Critical: ระบบใช้งานไม่ได้, data loss, security breach
- High: Feature หลักเสีย, no workaround
- Medium: Feature รองเสีย, มี workaround
- Low: Cosmetic, minor inconvenience

C. Priority:

- P0: Fix ทันที, block release
- P1: Fix ใน sprint นี้
- P2: Fix sprint หน้า
- P3: Fix เมื่อมีเวลา

ตัวอย่างการจัดหมวดหมู่:
Bug: SQL Injection in search
→ Type: Security
→ Severity: Critical (data breach possible)
→ Priority: P0 (fix immediately)

Bug: Misspelled text in help page
→ Type: UI/UX
→ Severity: Low (doesn't affect functionality)
→ Priority: P3 (fix when free)

Bug: Can borrow more than 5 books
→ Type: Business Logic
→ Severity: High (violates business rule)
→ Priority: P1 (fix this sprint)

Part 3: Writing Bug Reports
3.1 Bug Report Template
📝 ใช้ Template นี้สำหรับ Bug Report ทุก bug:

```
# Bug Report #[NUMBER]

## Title
[Concise, descriptive title]

## Environment
- **Browser:** Chrome 119.0
- **OS:** Windows 11
- **Build:** v1.0.0-buggy
- **Date:** 2024-xx-xx

## Severity & Priority
- **Severity:** [Critical/High/Medium/Low]
- **Priority:** [P0/P1/P2/P3]
- **Type:** [Security/Business Logic/Validation/etc.]

## Description
[Clear description of what's wrong]

## Steps to Reproduce
1. [First step]
2. [Second step]
3. [Third step]
...

## Expected Result
[What should happen]

## Actual Result
[What actually happens]

## Screenshots/Videos
[Attach if applicable]

## Additional Information
- **Reproducibility:** Always / Sometimes / Rare
- **Workaround:** [If any]
- **Related Bugs:** #[number]
- **Notes:** [Any other relevant info]
```

3.2 ตัวอย่าง Bug Report ที่ดี

```
# Bug Report #001

## Title
SQL Injection vulnerability in book search allows database access

## Environment
- **Browser:** Chrome 119.0.6045.159
- **OS:** macOS 14.1
```

- **Build:** v1.0.0-buggy
- **Date:** 2024-12-10

Severity & Priority
- **Severity:** Critical
- **Priority:** P0
- **Type:** Security

Description
The book search functionality is vulnerable to SQL injection attacks.
An attacker can inject SQL commands through the search input field,
potentially accessing or modifying the entire database.

Steps to Reproduce
1. Navigate to http://localhost:3000/search
2. In the search box, enter: `' OR '1'='1' --`
3. Click Search button
4. Observe the results

Expected Result
- System should sanitize input and show error message
- No SQL injection should be possible
- Only matching books should be returned

Actual Result
- All books in the database are returned
- SQL query is executed without sanitization
- Console shows the raw SQL query with injected code
- Database is accessible for manipulation

Screenshots
[Screenshot showing all books returned]
[Screenshot of browser console showing SQL error]

Additional Information
- **Reproducibility:** Always (100%)
- **Workaround:** None - this is a critical security flaw
- **Security Impact:**
  - Can view all database contents
  - Can potentially modify/delete data
  - Can extract sensitive user information
- **Affected Endpoints:**
  - GET /api/books/search
  - POST /api/books/advanced-search
- **Related Bugs:** None
- **Notes:**
  - Tested with multiple SQL injection payloads
  - All were successful
  - Need to implement parameterized queries
  - Need to add input validation

3.3 ตัวอย่าง Bug Report แบบต่าง ๆ

Example 1: Business Logic Bug

```
# Bug Report #002

## Title
Users can borrow more than 5 books simultaneously

## Severity & Priority
- **Severity:** High
- **Priority:** P1
- **Type:** Business Logic

## Steps to Reproduce
1. Login as a regular member
2. Go to any book detail page
3. Click "Borrow" button 6 times on different books
4. Check "My Books" page

## Expected Result
System should only allow borrowing maximum 5 books
Error message should appear when attempting to borrow 6th book

## Actual Result
All 6 books are borrowed successfully
No error message is displayed
Counter shows "6/5 books borrowed"
```

Example 2: UI/UX Bug

```
# Bug Report #003

## Title
Cancel button appears before Save button (incorrect order)

## Severity & Priority
- **Severity:** Low
- **Priority:** P3
- **Type:** UI/UX

## Description
On the "Edit Profile" form, the Cancel button is positioned
to the left of the Save button, which violates UX conventions
and may cause users to accidentally cancel instead of save.

## Expected Result
Save button should be on the right (primary action)
Cancel button should be on the left (secondary action)

## Actual Result
Buttons are reversed: [Cancel] [Save]
```

Assignment (Homework) - **ส่งภายใน 1 สัปดาห์**

Requirements:

1. หา bugs อย่างน้อย 10 bugs จากระบบ Library Management
2. ต้องมีอย่างน้อย:
    - 1 Security bug
    - 2 Business Logic bugs
    - 2 Validation bugs
    - 5 UI/UX bugs หรือ bugs อื่นๆ
3. เขียน Bug Report สำหรับแต่ละ bug ตาม template
4. จัดหมวดหมู่ bugs (Type, Severity, Priority)
5. สร้างตาราง summary ของ bugs ทั้งหมด

Deliverables:

- bugs-week1.md - รวม bug reports ทั้งหมด
- bugs-summary.xlsx - Excel file สรุป bugs
- อัพโหลดไปที่ GitHub repository ส่วนตัว

Grading Criteria:

- Quality of bug reports (40%)
- Variety of bug types found (30%)
- Correct severity/priority classification (20%)
- Documentation quality (10%)

💡 Tips for Bug Hunting

Where to Look:

1. Input fields - ลอง invalid data, special characters
2. Boundaries - ลองค่า min/max, zero, negative
3. Authentication - ลอง access โดยไม่ login
4. State changes - ลองกด button หลายครั้ง
5. Error handling - ดูว่า error messages เหมาะสมไหม

Good Tester Mindset:

- "What could go wrong?"
- "What if I do this differently?"
- "What happens if I do the opposite?"
- "Can I break this?"
- "Is this secure?"

Documentation Best Practices:

- ชัดเจน (Clear)
- กระชับ (Concise)
- ครบถ้วน (Complete)
- สามารถ reproduce ได้ (Reproducible)
- มี evidence (screenshots/logs)

สรุปสัปดาห์ที่ 1
สิ่งที่เรียนรู้:

- ความสำคัญของ Software Testing
- ความแตกต่างระหว่าง SDLC และ STLC
- Cost of Quality และ Cost of Bugs
- Testing vs Debugging vs QA
- Software Defects และการจัดหมวดหมู่
- การเขียน Bug Report ที่ดี
- Bug hunting techniques

ทักษะที่ได้:

- Setup development environment
- Exploratory testing
- Writing professional bug reports
- Bug classification (Type, Severity, Priority)
- Using browser developer tools

สัปดาห์หน้า (Week 2):

- Software Quality Attributes (ISO 25010)
- Quality Metrics (Defect Density, Coverage, DRE)
- การประเมินคุณภาพระบบ
- การคำนวณ metrics จากข้อมูลจริง

📚 Additional Resources
Articles:

- ISTQB Foundation Syllabus
- Software Testing Help
- Ministry of Testing

Books:

- "Software Testing" by Ron Patton
- "Lessons Learned in Software Testing" by Cem Kaner
- "The Art of Software Testing" by Glenford Myers

Tools Documentation:

- Node.js Docs
- Chrome DevTools
- Git Handbook

O]O