Software Testing

Week 11

ดัดแปลงจาก slides ของหนังสือ Software Engineering [1]

หัวข้อที่จะศึกษา

- Development testing
- Test-driven development
- Release testing
- User testing

Software reuse

- การทดสอบมีจุดมุ่งหมายเพื่อแสดงให้เห็นว่าโปรแกรมทำในสิ่งที่เราตั้งใจและช่วยให้ค้นพบข้อบกพร่องของโปรแกรมก่อนที่จะนำมาใช้งาน
- การทดสอบซอฟต์แวร์เป็นเรียกใช้โปรแกรมจริงโดยใช้ข้อมูลเทียม
- ในการทดสอบ เราตรวจสอบผลลัพธ์ของการทดสอบเพื่อหาข้อผิดพลาด ความผิดปกติต่าง ๆ หรือข้อมูลเกี่ยวกับ non-functional ของโปรแกรม
- การทดสอบ ทำเพื่อหาข้อผิดพลาด **ไม่ใช่**ทำเพื่อที่จะบอกว่า โปรแกรมของเราไม่มีข้อผิดพลาด

Program testing goals

- เพื่อแสดงให้นักพัฒนาซอฟต์แวร์และลูกค้าเห็นว่าซอฟต์แวร์ตรงกับ software requirement
  - สำหรับซอฟต์แวร์ที่ลูกค้าออกแบบเอง ควรมีการทดสอบอย่างน้อยหนึ่งข้อสำหรับแต่ละ requirement
  - สำหรับผลิตภัณฑ์ซอฟต์แวร์ทั่วไป ควรมีการทดสอบคุณลักษณะระบบทั้งหมด รวมถึง combination ของคุณลักษณะเหล่านั้น
- เพื่อค้นหาสถานการณ์ที่ซอฟต์แวร์ไม่ถูกต้องไม่พึงประสงค์หรือไม่เป็นไปตาม software requirement
  - เพื่อขจัดพฤติกรรมที่ไม่พึงประสงค์ เช่น ระบบล่ม, การโต้ตอบที่ไม่พึงประสงค์กับระบบอื่น, การคำนวณที่ไม่ถูกต้อง และความเสียหายของข้อมูล เป็นต้น

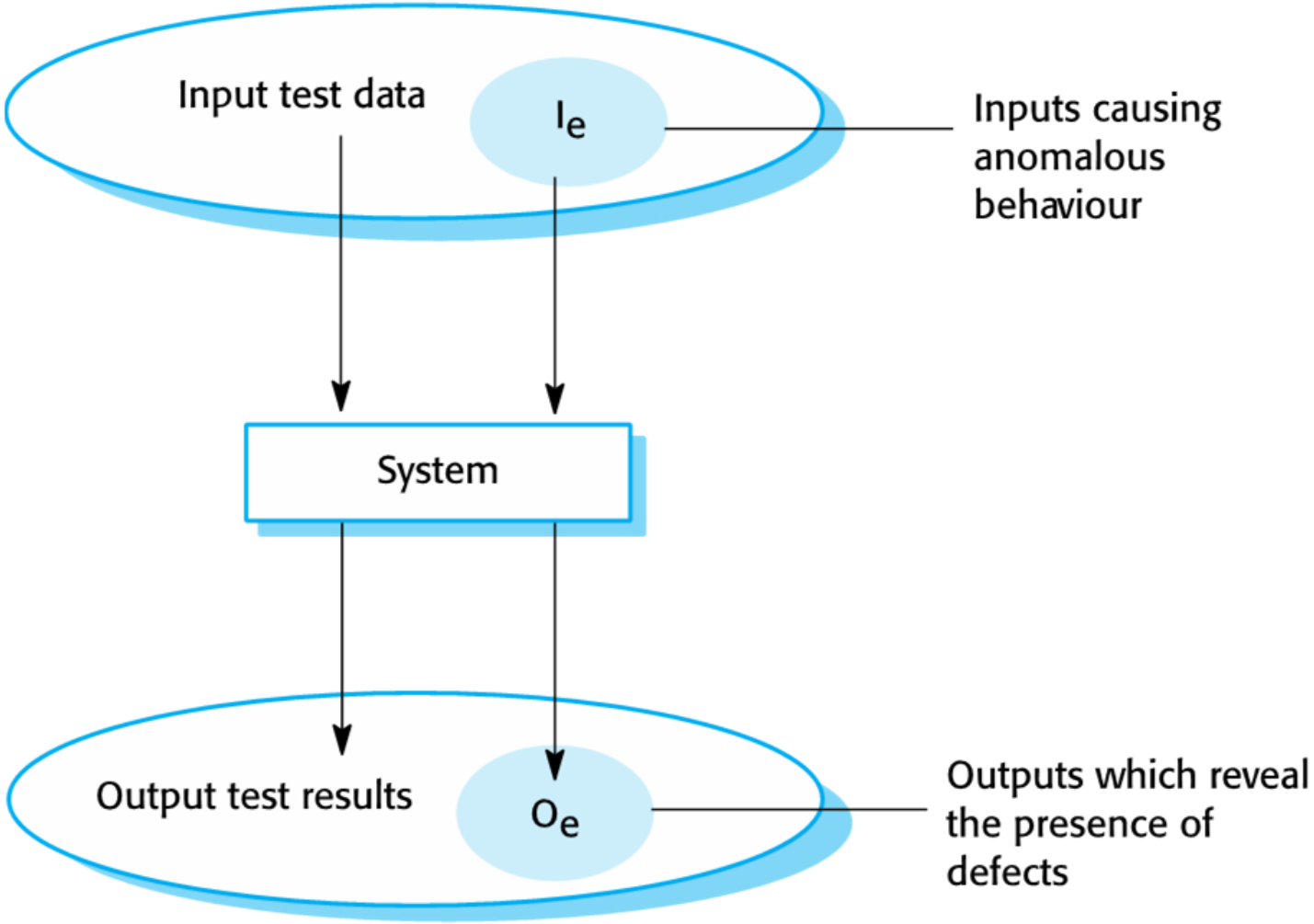
Validation and defect testing

- เป้าหมายแรกนำคือการทดสอบความถูกต้อง
  - คาดว่าระบบจะทำงานได้อย่างถูกต้องโดยใช้ test case ที่กำหนดซึ่งสะท้อนถึงการใช้งานที่คาดไว้ของระบบ
- เป้าหมายที่สองคือการทดสอบข้อบกพร่อง
  - test case ถูกออกแบบมาเพื่อแสดงข้อบกพร่อง
  - test case ในการทดสอบข้อบกพร่องอาจทำให้เกิดความคลุมเครือและไม่จำเป็นต้องสะท้อนถึงระบบที่ใช้ตามปกติ

Testing process goals

- การทดสอบความถูกต้อง
  - เพื่อแสดงให้ผู้พัฒนาและลูกค้าระบบเห็นว่าซอฟต์แวร์มีคุณสมบัติตรงตามข้อกำหนด
  - ความสำเร็จในการทดสอบนี้คือ การแสดงให้เห็นว่าระบบทำงานตามที่ตั้งใจ
- การทดสอบความบกพร่อง
  - เพื่อค้นหาข้อบกพร่องหรือข้อบกพร่องในซอฟต์แวร์ที่มีพฤติกรรมไม่ถูกต้องหรือไม่เป็นไปตามข้อกำหนด
  - ความสำเร็จในการทดสอบนี้คือ การทดสอบที่ทำให้ระบบทำงานได้ไม่ถูกต้องและทำให้เกิดข้อบกพร่องในระบบ

An input-output model of program testing



Verification vs validation

- การยืนยัน (Verification):
  - "Are we building the product right".
  - ซอฟต์แวร์ควรเป็นไปตามข้อกำหนด
- การตรวจสอบ (Validation):
  - "Are we building the right product".
  - ซอฟต์แวร์ควรทำในสิ่งที่ผู้ใช้ต้องการจริง ๆ

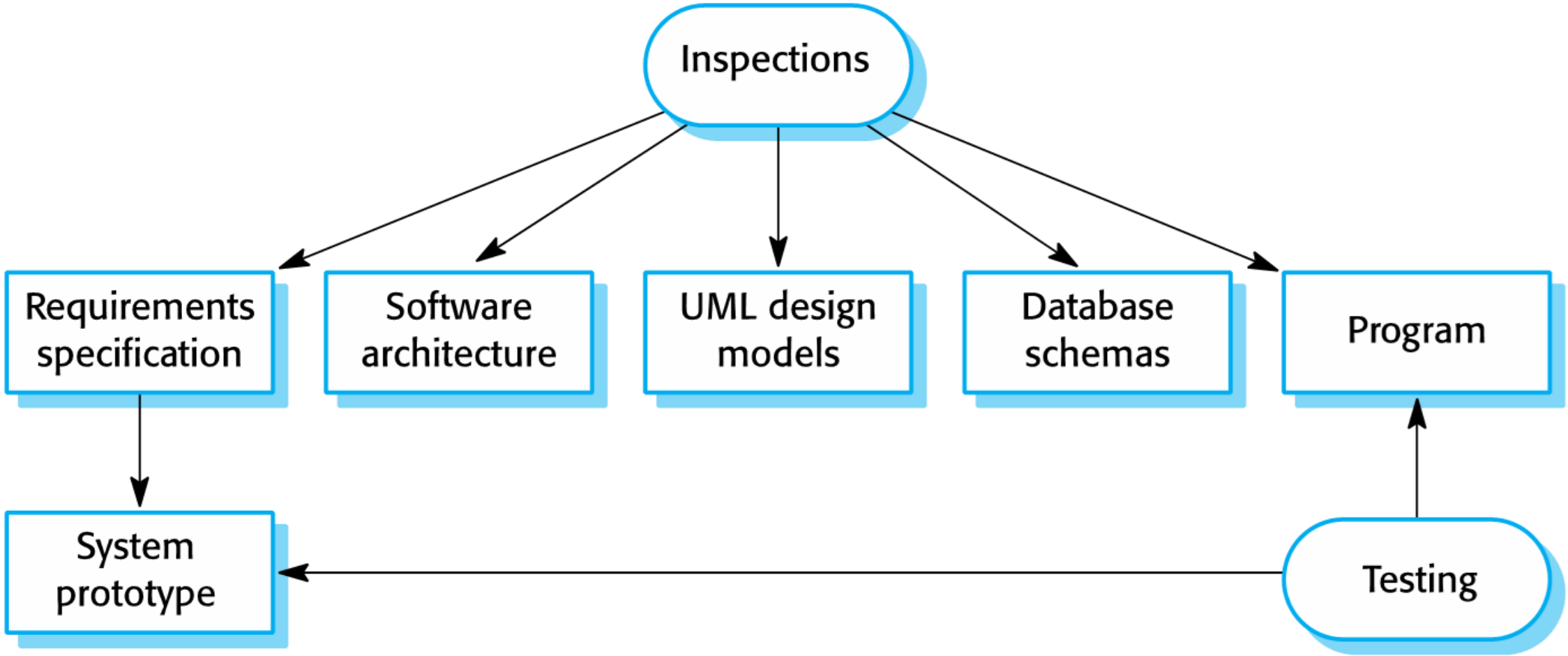
V & V confidence

- จุดมุ่งหมายของ V & V คือการสร้างความเชื่อมั่นว่าระบบนี้ ‘fit for purpose’.
- ขึ้นอยู่กับวัตถุประสงค์ของระบบ ความคาดหวังของผู้ใช้ และสภาพแวดล้อมทางการตลาด
  - วัตถุประสงค์ของซอฟต์แวร์
    - ระดับความเชื่อมั่นขึ้นอยู่กับความสำคัญของซอฟต์แวร์ต่อองค์กร
  - ความคาดหวังของผู้ใช้
    - ผู้ใช้อาจมีความคาดหวังต่ำในบางประเภทของซอฟต์แวร์
  - สภาพแวดล้อมทางการตลาด
    - การปล่อยผลิตภัณฑ์สู่ตลาดในช่วงต้นอาจมีความสำคัญมากกว่าการค้นหาข้อบกพร่องในโปรแกรม

Inspections and testing

- การตรวจสอบซอฟท์แวร์เพื่อค้นพบปัญหาในการวิเคราะห์ระบบแบบคงที่ (static verification)
  - อาจจะใช้ tool-based document and code analysis
- การทดสอบซอฟต์แวร์ที่เกี่ยวข้องกับทดสอบและการสังเกตพฤติกรรมของผลิตภัณฑ์ (dynamic verification)
  - ระบบถูกทดสอบด้วยข้อมูลเทียมและมีการสังเกตุพฤติกรรมการทำงาน

Inspections and testing



Software inspections

- อาจจะใช้คนตรวจสอบ source code โดยมีวัตถุประสงค์เพื่อค้นหาความผิดปกติและข้อบกพร่อง
- การตรวจสอบนี้ไม่จำเป็นต้อง run ระบบจึงอาจทำการตรวจสอบใช้ก่อนการใช้งานระบบจริงได้
- สามารถนำไปประยุกต์ใช้กับการนำเสนอระบบ (requirements, design, configuration data, test data, เป็นต้น).
- วิธีการนี้เป็นเทคนิคที่มีประสิทธิภาพสำหรับการค้นพบข้อผิดพลาดของโปรแกรม

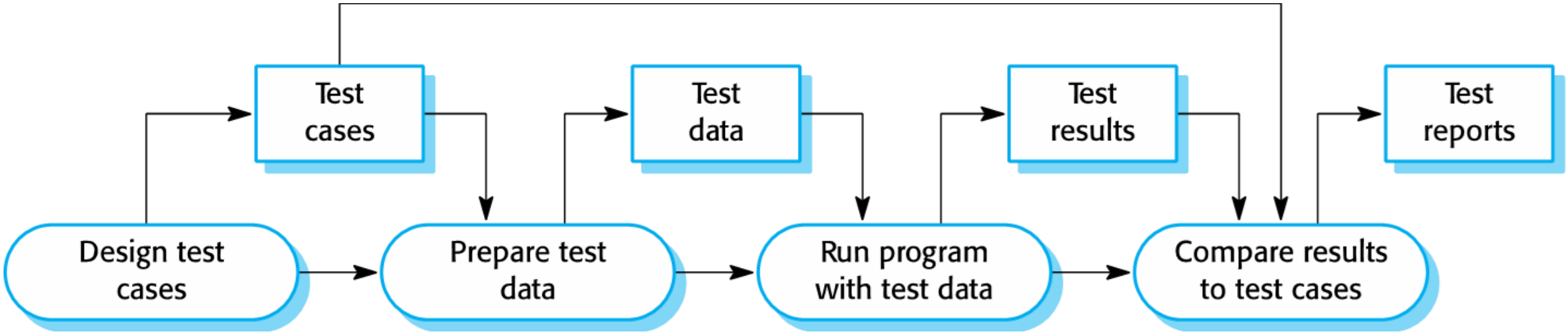
Advantages of inspections

- ในระหว่างการทดสอบ เรามักจะพบว่าข้อผิดพลาดหนึ่งสามารถปกปิด (ซ่อน) ข้อผิดพลาดอื่น ๆ ได้
  - แต่เนื่องจากการตรวจสอบเป็นกระบวนการแบบคงที่ เราจึงไม่จำเป็นต้องสนใจการเชื่อมโยงระหว่างข้อผิดพลาด
- สามารถตรวจสอบระบบเวอร์ชันที่ไม่สมบูรณ์ได้ (โดยไม่มีค่าใช้จ่ายเพิ่มเติม)
  - แต่หากโปรแกรมไม่สมบูรณ์เราจะต้องพัฒนา test harness ขึ้นมาโดยเฉพาะเพื่อทดสอบชิ้นส่วนที่พร้อมใช้งาน
- ในขณะค้นหาข้อบกพร่องของโปรแกรม เรายังสามารถพิจารณาคุณลักษณะอื่น ๆ ที่แสดงคุณภาพที่สูงขึ้นของโปรแกรม เช่น การเป็นไปตามมาตรฐาน ความเป็น portability และ maintainability

Inspections and testing

- การตรวจสอบ (Inspections) และการทดสอบ (testing) เป็นส่วนเสริมซึ่งกันและกัน และไม่ใช่เทคนิคที่มีความคัดค้านกันเอง
- ทั้งสองควรดำเนินการระหว่างกระบวนการ V & V
- การตรวจสอบ (Inspections) สามารถตรวจสอบความสอดคล้องกับข้อกำหนด แต่อาจจะไม่สอดคล้องกับความต้องการที่แท้จริงของลูกค้า
- การตรวจสอบไม่สามารถตรวจสอบคุณลักษณะที่เป็น non-functional เช่น ประสิทธิภาพ, การใช้งาน ฯลฯ

A model of the software testing process



Stages of testing

- Development testing
  - ระบบจะถูกทดสอบในระหว่างการพัฒนา เพื่อค้นหาข้อผิดพลาดและข้อบกพร่อง
- Release testing,
  - ทำโดยทีมทดสอบที่แยกจากทีมพัฒนา จะทดสอบเวอร์ชันที่สมบูรณ์ของระบบก่อนปล่อยออกสู่ผู้ใช้
- User testing
  - ผู้ใช้ (หรือผู้ที่อาจเป็นผู้ใช้ระบบ) ทำการทดสอบระบบในสภาพแวดล้อมของตนเอง

Development testing

Development testing

- Development testing ประกอบด้วยกิจกรรมการทดสอบทั้งหมดที่ดำเนินการโดยทีมพัฒนาระบบ
- Unit testing
  - Unit testing แต่ละหน่วยย่อยโปรแกรมหรือคลาสของวัตถุจะได้รับการทดสอบ
  - Unit testing ควรเน้นการทดสอบฟังก์ชันการทำงานของ object หรือ method
- Component testing
  - เป็นการทดสอบชิ้นส่วนซึ่งมีการรวมหน่วยต่าง ๆ ไว้หลายชิ้นเพื่อสร้าง composite components
  - การทดสอบส่วนประกอบควรมุ่งเน้นไปที่การทดสอบ interface ระหว่าง components
- System testing
  - ส่วนประกอบบางส่วนหรือทั้งหมดของระบบถูกรวมเข้าด้วยกันและทดสอบระบบโดยรวม
  - การทดสอบระบบควรเน้นการทดสอบการโต้ตอบระหว่างส่วนประกอบต่าง ๆ

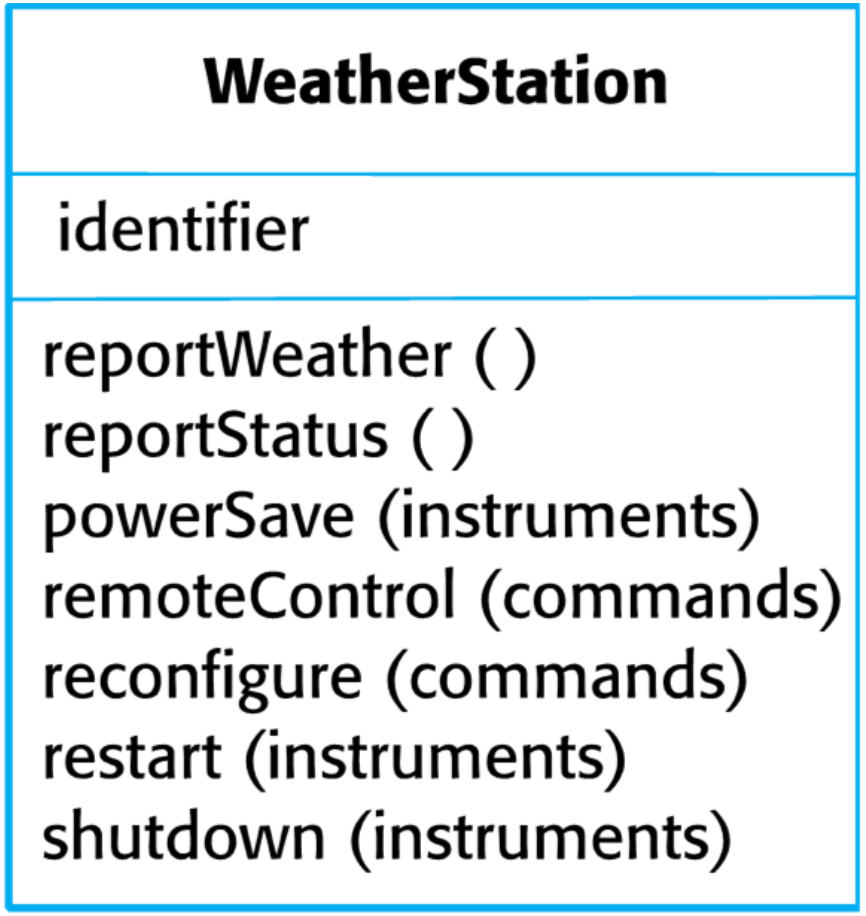
Unit testing

- Unit testing เป็นขั้นตอนการทดสอบส่วนประกอบแต่ละชิ้นโดยแยกออกจากกัน
- เป็นกระบวนการทดสอบข้อบกพร่อง
- Unit อาจหมายถึง
  - Function หรือ method ภายในวัตถุ
  - Object classes ที่มี attributes และ methods อยู่ภายใน
  - Composite components ที่มี interface ที่กำหนดไว้ เพื่อเข้าถึงฟังก์ชันการทำงานของพวกมัน

Object class testing

- ครอบคลุมการทดสอบทั้งหมดของ class
  - ทดสอบการทำงานทั้งหมดที่เกี่ยวข้องกับ object
  - การตั้งค่าและการสอบถามคุณลักษณะทั้งหมดของ object
  - การใช้ object ในทุกสภาวะที่เป็นไปได้
- การสืบทอดคลาส (Inheritance) ทำให้การออกแบบการทดสอบในวัตถุทำได้ยากลำบากยิ่งขึ้น เนื่องจากข้อมูลที่จะทดสอบอาจจะไม่ใช่ข้อมูล localized ของวัตถุนั้น

Case study: The weather station object interface



Weather station testing

- ต้องกำหนด test cases สำหรับ reportWeather, calibrate, test, startup และ shutdown
- ใช้แบบจำลอง state model เพื่อระบุลำดับของการเปลี่ยนสถานะที่จะทดสอบ และลำดับเหตุการณ์จะทำให้เกิดการเปลี่ยนเหล่านั้น
- ตัวอย่างเช่น:
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Automated testing

- ทุกโอกาสที่เป็นไปได้ การทดสอบ unit testing ควรเป็นแบบอัตโนมัติ เพื่อให้การทดสอบดำเนินไปและตรวจสอบโดยไม่มีการแทรกแซงโดยมนุษย์
- ในการทดสอบ unit testing โดยอัตโนมัติ เราจะใช้ test automation framework (เช่น JUnit) เพื่อเขียนและรันการทดสอบโปรแกรมให้เรา
- Unit testing frameworks มี test class ทั่วไปที่เราสามารถ extend เพื่อสร้าง test case จากนั้นพวกมันจะเรียกใช้ (run) การทดสอบทั้งหมดที่เราเขียนขึ้น และรายงานผลการทดสอบออกมา

Automated test components

- ส่วนการติดตั้ง (setup part)
  - จะเริ่มต้นระบบด้วยกรณีทดสอบ ได้แก่ อินพุตและเอาท์พุตที่คาดหวัง
- ส่วนการเรียกใช้ (calling part)
  - จะเรียกใช้งาน object หรือ method ที่จะทดสอบ
- ส่วนยืนยันผลการเปรียบเทียบ (assertion part)
  - จะเปรียบเทียบผลของการเรียกใช้งานกับผลลัพธ์ที่คาดหวังไว้
  - ถ้าการการเปรียบเทียบยืนยันว่าเป็นความจริงการทดสอบนี้ประสบความสำเร็จถ้าเป็นเท็จแล้วก็จะล้มเหลว

Choosing unit test cases

- เมื่อใช้ test cases ตามที่กำหนดไว้ component ที่กำลังทดสอบจะต้องทำในสิ่งที่คาดหวัง
- หากมีข้อบกพร่องใด ๆ ใน component ควรถูกตรวจพบโดย test cases
- การทดสอบ unit test มี 2 กรณี
  - กรณีแรกควรสะท้อนถึงการทำงานโดยปกติของโปรแกรม และควรแสดงให้เห็นว่าส่วนประกอบทำงานได้ตามที่คาดไว้
  - กรณีที่สองจะขึ้นอยู่กับประสบการณ์การทดสอบ
    - โดยทั่วไป ควรใช้อินพุตที่ผิดปกติเพื่อตรวจสอบว่าได้รับการประมวลผลอย่างถูกต้องโดย component และไม่ทำให้ component นั้นเสียหาย

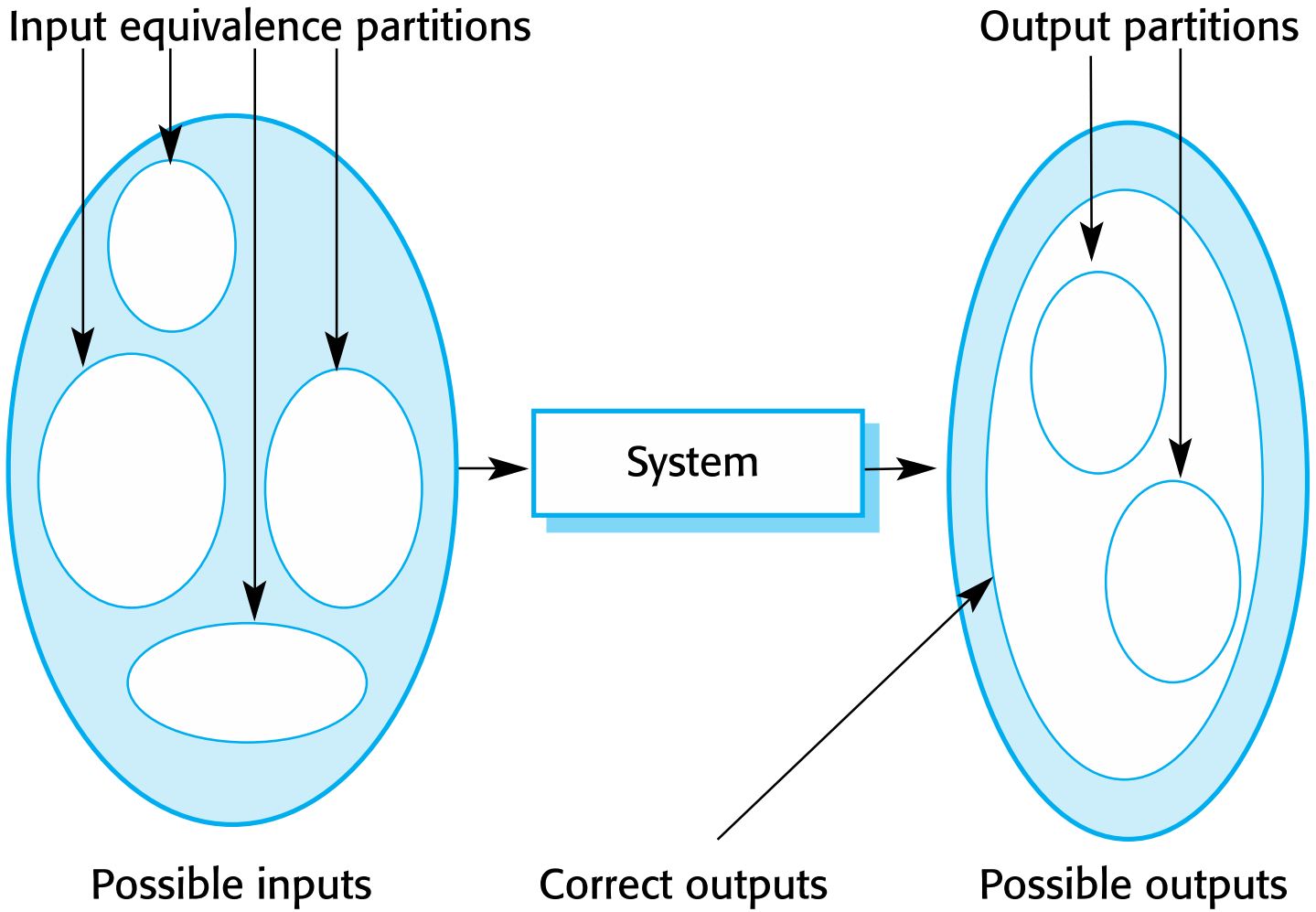
Testing strategies

- Partition testing
  - อินพุตที่มีลักษณะเหมือนกัน ควรได้รับการประมวลผลในลักษณะเดียวกัน
- Guideline-based testing
  - การทดสอบตามเกณฑ์ จะใช้หลักเกณฑ์ในการทดสอบเพื่อเลือก test case
  - Guideline เหล่านี้มักได้จากประสบการณ์เกี่ยวกับข้อผิดพลาดประเภทต่าง ๆ ที่โปรแกรมเมอร์มักเจอและใช้แก้ปัญหาในการพัฒนา components

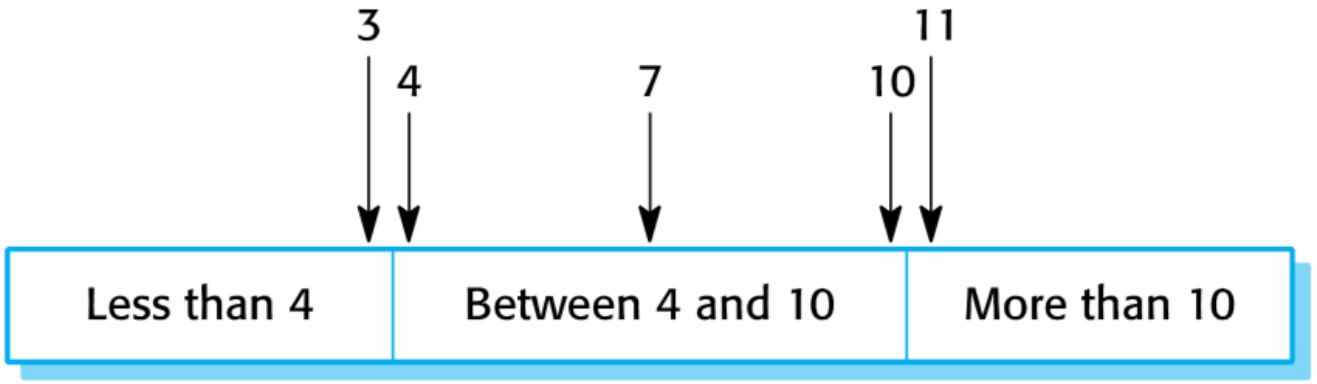
Partition testing

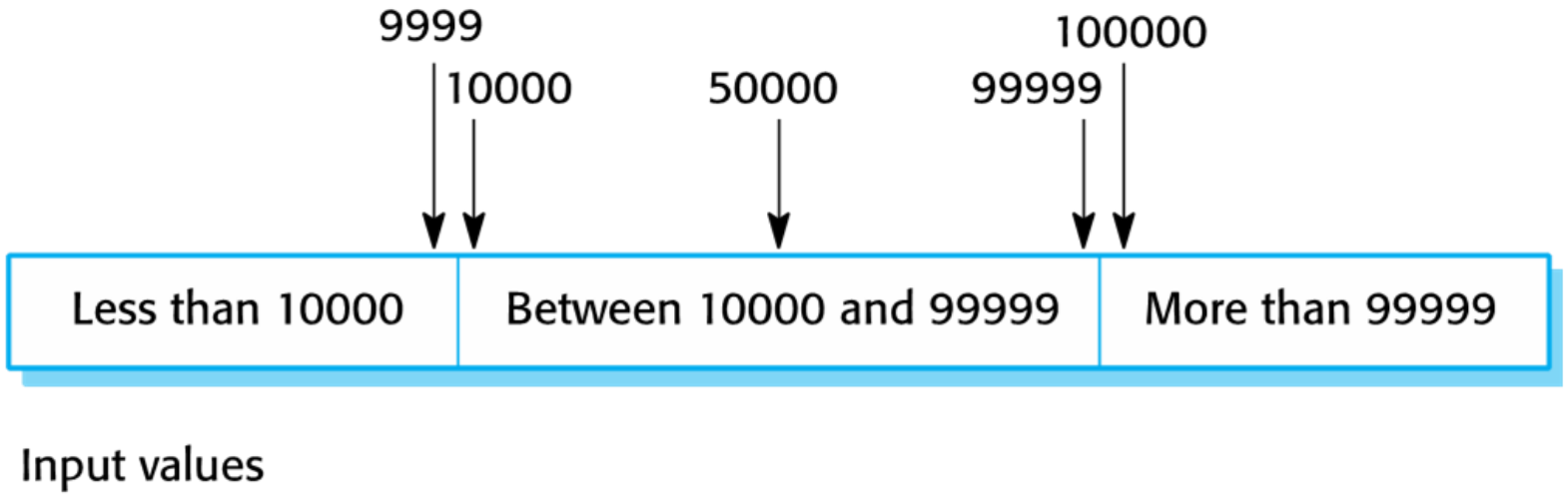
- ข้อมูล input และ output มักจะถูกนำไปใช้ในหลาย ๆ class และสมาชิกใน class ทั้งหมดมีความเกี่ยวข้องกัน
- แต่ละ class เหล่านี้เป็นพาร์ติชันหรือโดเมนที่เท่าเทียมกัน ซึ่งโปรแกรมทำงานในลักษณะที่เทียบเท่ากันสำหรับสมาชิกทั้งหมดของ class
- ควรเลือก test case จากแต่ละพาร์ติชัน

Equivalence partitioning



Equivalence partitions





Testing guidelines (sequences)

- ทดสอบซอฟต์แวร์ด้วยลำดับ (sequences) ที่มีเพียงค่าเดียว
- ใช้ลำดับขนาดต่าง ๆ กัน ในการทดสอบที่แตกต่างกัน
- ทำการทดสอบเพื่อให้สามารถเข้าถึงองค์ประกอบลำดับแรกกลางและสุดท้ายของลำดับได้
- ทดสอบด้วยลำดับของความยาวเป็นศูนย์

General testing guidelines

- เลือกอินพุตที่บังคับให้ระบบสร้างข้อความแสดงข้อผิดพลาดทั้งหมด
- ออกแบบอินพุทที่ทำให้บัฟเฟอร์อินพุตเกิดการ overflow
- ใช้งานอินพุทเดียวกันหรือ series ของอินพุตซ้ำหลายๆ ครั้ง
- บังคับให้ component สร้างผลลัพธ์ที่ไม่ถูกต้องขึ้นมา
- บังคับให้ผลการคำนวณมีขนาดใหญ่เกินไปหรือเล็กเกินไป

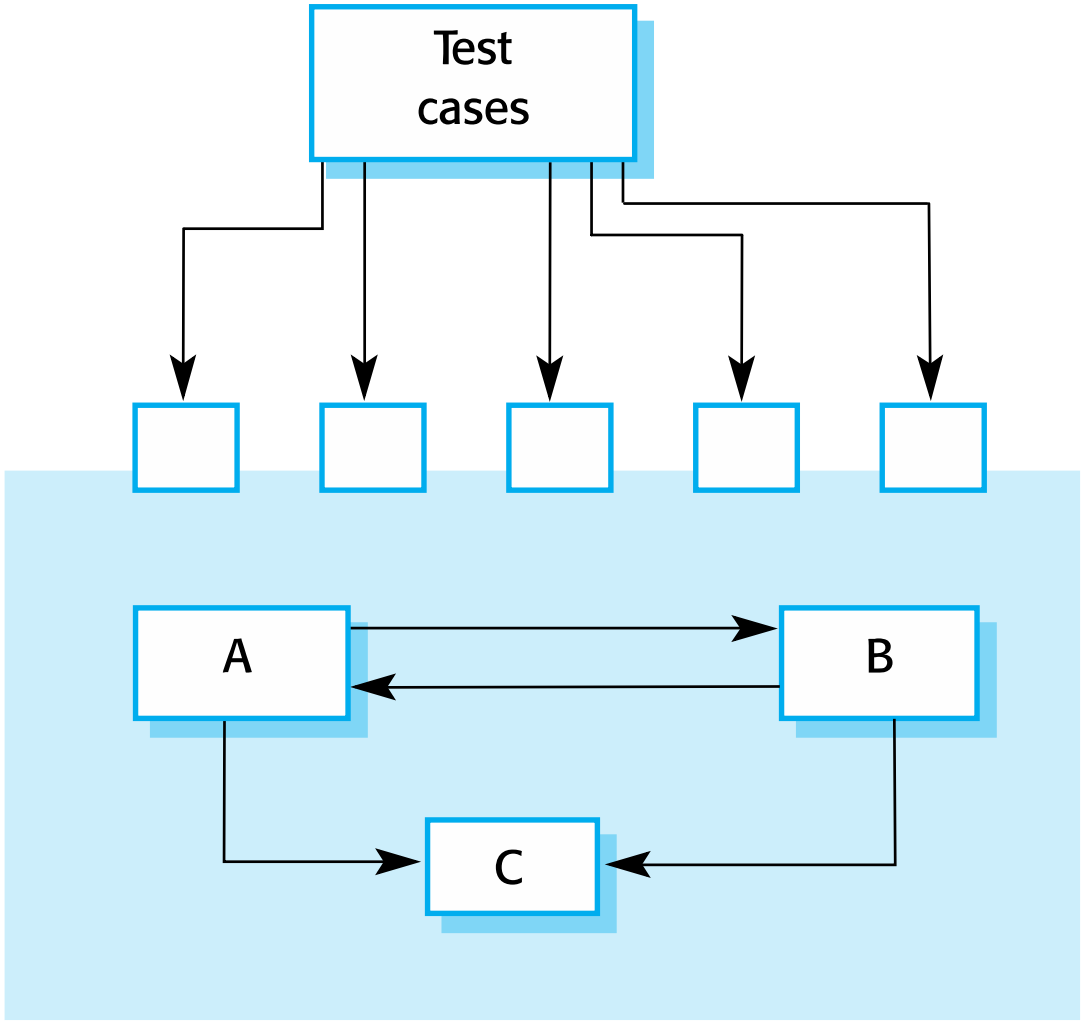
Component testing

- Software components มักประกอบขึ้นจาก components ที่สร้างจาก object จำนวนหนึ่งที่มีการโต้ตอบซึ่งกันและกันในรูปแบบต่าง ๆ
  - ตัวอย่างเช่นในระบบ weather station นั้น reconfiguration component ประกอบด้วย object ที่เกี่ยวข้องกับแต่ละด้านของ reconfiguration
- เราสามารถเข้าถึงฟังก์ชันการทำงานของ object เหล่านี้ผ่าน interface ของ component ที่กำหนด
- การทดสอบ composite component ควรเน้นที่การแสดงให้เห็นว่า interface ของส่วนประกอบทำงานได้ตามข้อกำหนด
  - ทั้งนี้อยู่บนข้อสันนิษฐานว่ามีการทดสอบ unit test ของแต่ละ object ภายใน component เหล่านั้นมาเสร็จสิ้นเป็นที่เรียบร้อยแล้ว

Interface testing

- วัตถุประสงค์คือเพื่อตรวจจับข้อผิดพลาดอันเนื่องมาจากข้อผิดพลาดของ interface หรือทดสอบข้อสันนิษฐานเกี่ยวกับ interface นั้น
- ชนิดของ interface
  - Parameter interfaces เป็นข้อมูลที่ส่งผ่านจาก method หรือ procedure หนึ่งไปยังอีก procedure หนึ่ง
  - Shared memory interfaces เป็นหน่วยความจำที่ใช้ร่วมกันระหว่าง procedures หรือ functions
  - Procedural interfaces เป็น sub-system ที่ encapsulate ชุดของ procedures ที่จะเรียกโดย sub-system อื่น ๆ
  - Message passing interfaces เป็น sub-system ที่ขอบริการจาก sub-system อื่น ๆ

Interface testing



Interface errors

- ใช้งาน interface ผิดวิธี
  - Component เรียกใช้ component อื่นและทำให้เกิดข้อผิดพลาดในการใช้ interface เช่น พารามิเตอร์ผิดพลาด
- ความเข้าใจผิดเกี่ยวกับ interface
  - Component ที่เป็นผู้เรียกมีข้อสันนิษฐานเกี่ยวกับลักษณะการทำงานของ component ที่เรียกอย่างไม่ถูกต้อง
- ข้อผิดพลาดด้านเวลา
  - Component ผู้เรียกและผู้ถูกเรียกทำงานด้วยความเร็วที่แตกต่างกันและใช้งานข้อมูลที่ไม่อัพเดต (ตามวงรอบการทำงาน)

Interface testing guidelines

- ออกแบบ test เพื่อให้พารามิเตอร์ของ procedure ที่ถูกเรียกอยู่ในช่วงปลายสุดของ range
- ทดสอบ pointer เสมอโดยใช้ค่า null
- ออกแบบ test ซึ่งทำให้ component ทำงานไม่ผ่าน
- ใช้ stress testing ในระบบส่งข้อความ
- ในระบบหน่วยความจำร่วมกัน (shared memory systems) ให้เปลี่ยนแปลงลำดับการทำงานของ component

System testing

- การทดสอบระบบในระหว่างการพัฒนาเกี่ยวข้องกับการรวม component เพื่อสร้างเวอร์ชันของระบบแล้วจึงทดสอบระบบบรวม
- สิ่งที่มุ่งเน้นในการทดสอบระบบคือ การทดสอบปฏิสัมพันธ์ระหว่าง component ต่างๆ
- การทดสอบระบบตรวจสอบว่า component สามารถทำงานร่วมกันได้อย่างถูกต้องและส่งข้อมูลที่ถูกต้องในเวลาที่เหมาะสมผ่าน interface ของตน
- การทดสอบระบบทดสอบพฤติกรรมที่เกิดขึ้นใหม่ของระบบ ซึ่งต่างไปจาก component เดี่ยว ๆ

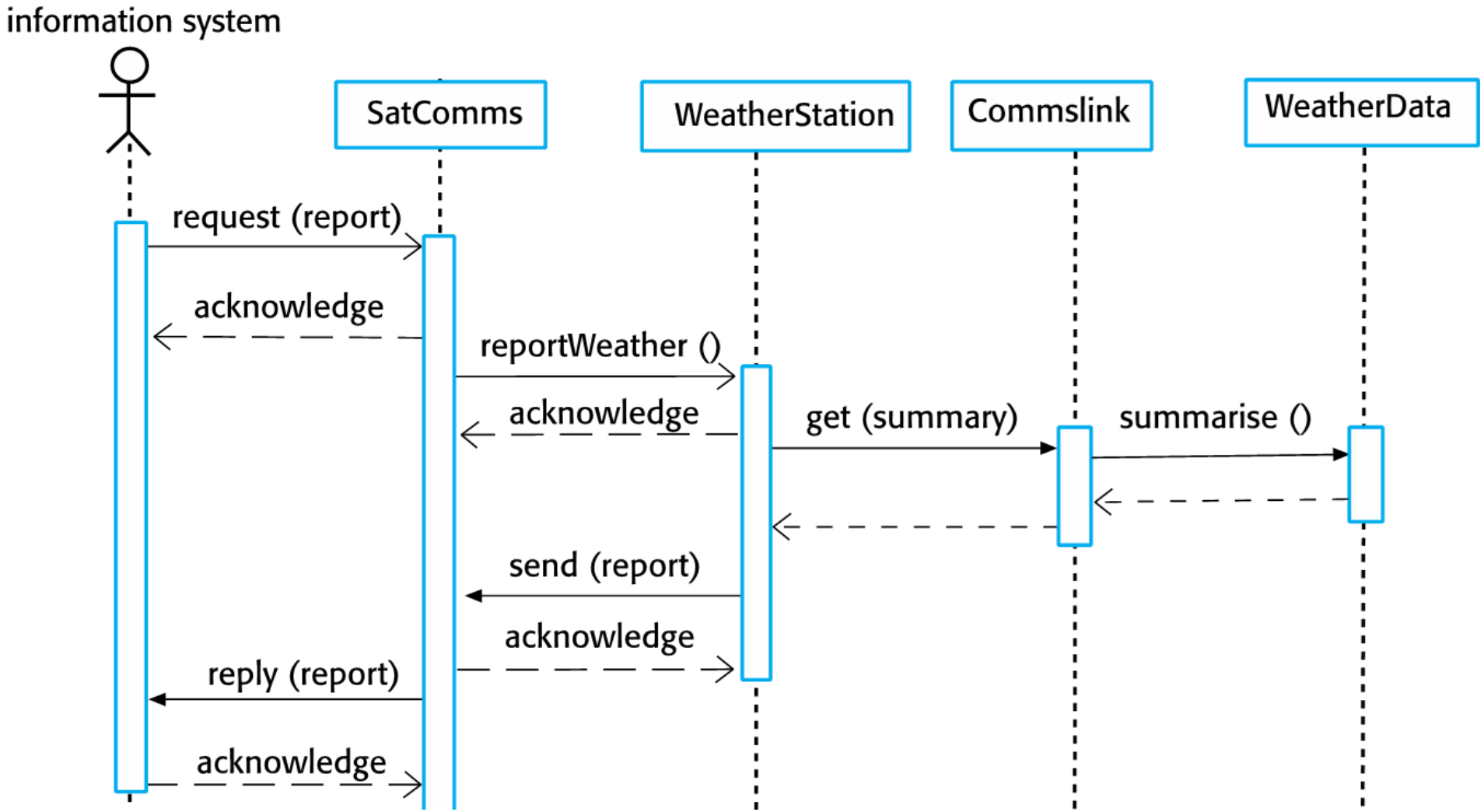
System and component testing

- ในระหว่างการทดสอบระบบนั้น component ที่พัฒนาขึ้นโดยเฉพาะ รวมทั้ง component ที่นำมา reuse และระบบ off-the-shelf อาจรวมเข้าด้วยกัน
  - เมื่อทดสอบส่วนย่อยเสร็จ เราอาจถือได้ว่าระบบที่สมบูรณ์ได้รับการทดสอบแล้ว
- Component ที่พัฒนาขึ้นโดยสมาชิกในทีมหรือทีมย่อยต่าง ๆ อาจรวมอยู่ในขั้นตอนนี้ การทดสอบระบบใช้ทรัพยากรส่วนรวมมากกว่าแต่ละขั้นตอน
  - ในบางบริษัท การทดสอบระบบอาจเกี่ยวข้องกับทีมทดสอบที่แยกออกไปต่างหาก โดยไม่มีส่วนเกี่ยวข้องกับนักออกแบบและนักเขียนโปรแกรม

Use-case testing

- Use-case ที่ถูกพัฒนาขึ้นมาเพื่อระบุปฏิสัมพันธ์ในระบบ สามารถใช้เป็นพื้นฐานสำหรับการออกแบบการทดสอบระบบ
- แต่ละ Use-case มักจะเกี่ยวข้องกับองค์ประกอบของระบบหลายอย่าง ดังนั้น ในการทดสอบระบบต้องบังคับให้ปฏิสัมพันธ์เหล่านี้เกิดขึ้นให้ครบถ้วน
- Sequence diagram ที่เชื่อมโยงกับ use-case จะอธิบายถึงส่วนประกอบและปฏิสัมพันธ์ที่จะต้องได้รับการทดสอบ

Collect weather data sequence chart



Test cases derived from sequence diagram

- อินพุทคำขอสำหรับรายงานควรมี acknowledgement ที่เกี่ยวข้อง (รายงานควรได้รับการส่งกลับมายังการร้องขอ)
  - ควรสร้างข้อมูลสรุปที่สามารถใช้เพื่อตรวจสอบว่ามีการจัดระเบียบรายงานอย่างถูกต้อง
- คำขอทางด้านอินพุทสำหรับรายงานถูกส่งไปยัง WeatherStation ส่งผลให้รายงานสรุปถูกสร้างขึ้น
  - สามารถทดสอบโดยการสร้างข้อมูลดิบที่สอดคล้องกับข้อมูลสรุปที่ได้เตรียมไว้

Testing policies

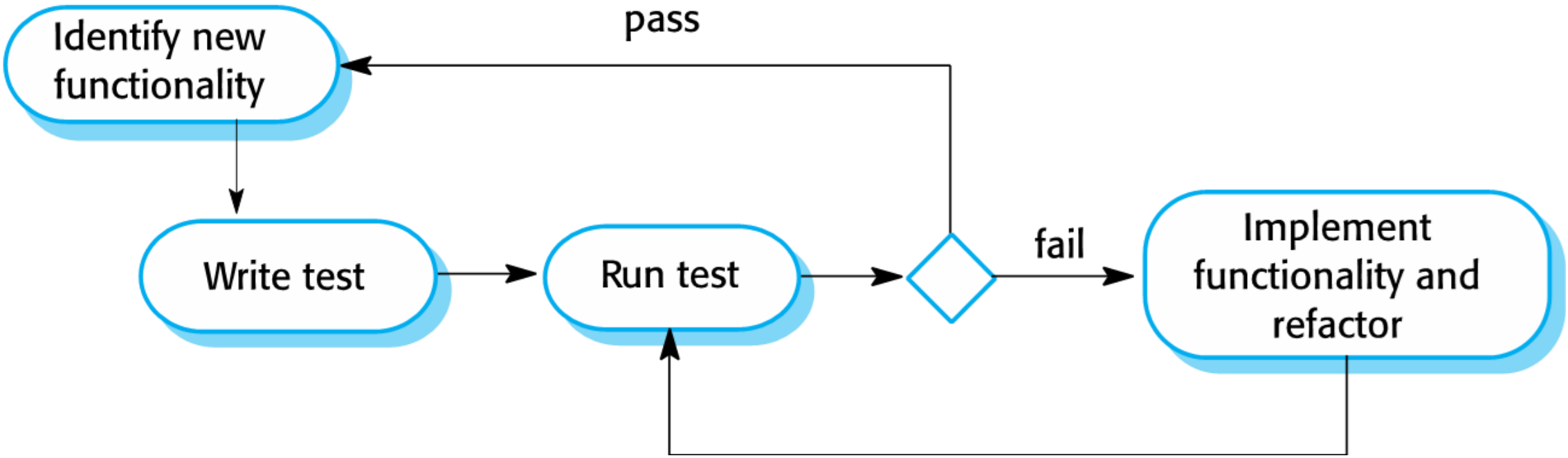
- เป็นไปไม่ได้ที่จะทำการทดสอบระบบได้อย่างครบถ้วนสมบูรณ์ ดังนั้นจึงอาจมีการดนโยบายในการทดสอบ ซึ่งกำหนดให้มีความครอบคลุมการทดสอบระบบเท่าที่จำเป็น
- ตัวอย่างของนโยบายการทดสอบ:
  - ควรทดสอบระบบทั้งหมดที่เข้าถึงได้ผ่านทางเมนู
  - ต้องมีการทดสอบชุดค่าผสมของฟังก์ชัน (เช่นการจัดรูปแบบข้อความ) ที่เข้าถึงได้ผ่านเมนูเดียวกัน
  - เมื่อมีการป้อนข้อมูลจากผู้ใช้ระบบจะต้องทดสอบฟังก์ชันทั้งหมดด้วยอินพุตที่ถูกต้องและไม่ถูกต้อง

Test-driven development

Test-driven development

- Test-driven development (TDD) เป็นแนวทางในการพัฒนาโปรแกรมซึ่งคุณจะทำการทดสอบและการพัฒนาโค้ดควบคู่หรือสลับกันไป
  - Test จะถูกเขียนขึ้นก่อนการเขียน code และ 'passing' การทดสอบจะเป็นตัวขับเคลื่อนที่สำคัญในการพัฒนา
- เราพัฒนา code ทีละน้อยพร้อมกับการ test สำหรับแต่ละ increment
  - ไม่ย้ายไปยัง increment ถัดไปจนกว่า code ที่ได้พัฒนาจะผ่านการ test
- TDD ถูกนำมาใช้เป็นส่วนหนึ่งของวิธีการแบบ agile เช่น Extreme Programming (XP)
  - อย่างไรก็ตามยังสามารถนำมาใช้ในกระบวนการพัฒนาแบบ plan-driven

Test-driven development



TDD process activities

- เริ่มต้นด้วยการระบุ increment ฟังก์ชันการทำงานที่จำเป็น
  - โดยปกติแล้วควรมีขนาดเล็กและสามารถใช้งานได้ในไม่กี่บรรทัด
- เขียนการทดสอบสำหรับฟังก์ชันนี้และให้การทดสอบนี้เป็นแบบอัตโนมัติ
- เรียกใช้การทดสอบพร้อมกับการทดสอบอื่น ๆ ทั้งหมดที่ได้รับการดำเนินการ
  - ในตอนแรกเรายังไม่ได้ implementation ดังนั้นการทดสอบใหม่จะล้มเหลว
- Implement ฟังก์ชันการทำงานและเรียกใช้การทดสอบอีกครั้ง
- เมื่อการทดสอบทั้งหมดทำงานได้สำเร็จ เราจะขยับไปยัง increment ต่อไป เพื่อ implement ฟังก์ชันต่อไป

Benefits of test-driven development

- ครอบคลุมโค้ด (Code coverage)
  - ทุกส่วนของโค้ดที่เขียนมีการทดสอบที่เกี่ยวข้องอย่างน้อยหนึ่งรายการ ดังนั้นโค้ดทั้งหมดจึงมีการทดสอบอย่างน้อยหนึ่งรายการ
- การทดสอบการถดถอย (Regression testing)
  - มีการพัฒนาชุดทดสอบการถดถอยแบบทวีคูณขึ้นเมื่อมีการพัฒนาโปรแกรม
- การแก้จุดบกพร่องได้ง่าย (Simplified debugging)
  - เมื่อการทดสอบล้มเหลวจะรู้ได้อย่างชัดเจนว่าปัญหาอยู่ที่ใด รหัสที่เขียนใหม่ต้องได้รับการตรวจสอบและแก้ไข
- เอกสารระบบ (System documentation)
  - การทดสอบตัวเองเป็นรูปแบบของเอกสารอธิบายถึงสิ่งที่โค้ดควรทำ

Regression testing

- การทดสอบการถดถอย (Regression testing) คือการทดสอบระบบเพื่อตรวจสอบว่าการเปลี่ยนแปลงไม่ได้ ‘broken’ code ก่อนหน้านี้
- ในขั้นตอนการทดสอบด้วยคน การทดสอบการถดถอยมักมีราคาแพง
  - แต่ด้วยการทดสอบแบบอัตโนมัติ จะทำได้ง่ายและตรงไปตรงมา
  - การทดสอบทั้งหมดจะถูกรันใหม่ทุกครั้งที่มีการเปลี่ยนแปลงโปรแกรม
- การทดสอบต้องทำงานได้ 'successfully' ก่อนที่จะมีการ commit code

Release testing

Release testing

- Release testing คือกระบวนการของการทดสอบการ release ระบบที่มีไว้สำหรับใช้กับภายนอกทีมพัฒนา
- เป้าหมายหลักของกระบวนการทดสอบ release test คือการโน้มน้าวผู้จัดหาระบบว่าดีพอสำหรับการใช้งาน
  - Release testing จึงต้องแสดงให้เห็นว่าระบบมีฟังก์ชันการทำงานที่ระบุ, มีประสิทธิภาพและความเชื่อถือได้ และไม่เกิดความผิดพลาดในระหว่างการใช้งานตามปกติ
- การทดสอบ release test เป็นกระบวนการทดสอบกล่องดำ (black-box) ซึ่งการทดสอบจะได้มาจากข้อกำหนดเฉพาะของระบบเท่านั้น

Release testing and system testing

- Release testing เป็นรูปแบบหนึ่งของ system testing
- ความแตกต่างที่สำคัญ:
  - ทีมที่แยกต่างหาก (ซึ่งไม่ได้มีส่วนร่วมในการพัฒนาระบบ) ควรเป็นผู้รับผิดชอบ release testing
  - การทดสอบระบบ (system testing) ซึ่งทำโดยทีมพัฒนา ควรเน้นการค้นหาข้อบกพร่องในระบบ (defect testing)
  - วัตถุประสงค์ของ release testing คือการตรวจสอบว่าระบบตรงกับความต้องการและดีพอสำหรับการใช้งานโดยผู้ใช้ภายนอก (validation testing)

Requirements based testing

- การทดสอบตามความต้องการ (Requirements based testing) เกี่ยวข้องกับการตรวจสอบความต้องการแต่ละอย่างและการพัฒนาแบบทดสอบหรือทดสอบระบบ
- ตัวอย่างความต้องการระบบ Mentcare:
  - หากรู้ว่าผู้ป่วยแพ้ยาชนิดใด ๆ แล้ว การสั่งยานั้นจะส่งผลให้มีข้อความแจ้งเตือนแก่ผู้ใช้ระบบ
  - หากผู้จ่ายยาเลือกที่จะไม่สนใจคำเตือน พวกเขาจะต้องให้เหตุผลว่าทำไมคำเตือนนี้จึงได้ถูกเพิกเฉย

Requirements tests

- ตั้งค่าประวัติผู้ป่วยที่ไม่มีอาการแพ้ยา กำหนดจ่ายยาสำหรับโรคภูมิแพ้ที่รู้จักกันทั่วไป ตรวจสอบว่าระบบจะไม่ออกข้อความแจ้งเตือน
- ตั้งค่าประวัติผู้ป่วยที่แพ้ยา กำหนดจ่ายยาที่ผู้ป่วยแพ้และตรวจสอบว่ามีคำเตือนออกโดยระบบหรือไม่
- บันทึกประวัติผู้ป่วยที่แพ้ยาตั้งแต่สองตัวขึ้นไป กำหนดยาทั้งสองตัวนี้แยกต่างหากและตรวจสอบว่ามีการออกคำเตือนที่ถูกต้องสำหรับยาแต่ละชนิด
- กำหนดยาสองตัวที่ผู้ป่วยแพ้ ตรวจสอบว่ามีการออกสองคำเตือนได้ถูกต้อง
- กำหนดให้มีการจ่ายยาที่ออกคำเตือนและลบล้างคำเตือนนั้น ตรวจสอบว่าระบบต้องการให้ผู้ใช้ให้ข้อมูลอธิบายว่าเหตุใดคำเตือนจึงถูกละเลย

Performance testing

- ส่วนหนึ่งของ release testing อาจเกี่ยวข้องกับการทดสอบคุณสมบัติอื่น ๆ ของระบบเช่น ประสิทธิภาพและความน่าเชื่อถือ
- การทดสอบควรจะสะท้อนถึงรูปแบบการใช้งานของระบบ
- การทดสอบประสิทธิภาพ (Performance tests) มักเกี่ยวข้องกับการวางแผนชุดทดสอบ เพื่อให้มีโหลดเพิ่มขึ้นอย่างต่อเนื่อง จนกว่าประสิทธิภาพของระบบจะไม่สามารถเป็นที่ยอมรับได้
- การทดสอบความเครียด (stress testing) เป็นรูปแบบหนึ่งของการทดสอบสมรรถนะ โดยการกำหนดให้ระบบมีการโอเวอร์โหลด เพื่อทดสอบพฤติกรรมความล้มเหลวของระบบ

User testing

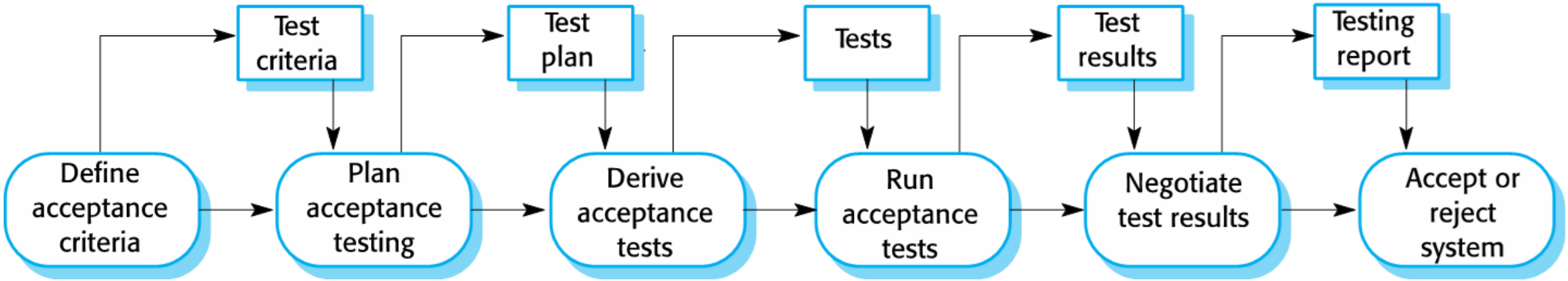
User testing

- User testing เป็นขั้นตอนในกระบวนการทดสอบ ซึ่งผู้ใช้หรือลูกค้าให้ข้อมูลและคำแนะนำในการทดสอบระบบ
- User testing เป็นสิ่งสำคัญแม้ว่าจะมีการทดสอบระบบและการทดสอบเวอร์ชันเต็มแล้วก็ตาม
- อิทธิพลจากสภาพแวดล้อมการทำงานของผู้ใช้ มักจะมีผลต่อ ความน่าเชื่อถือ, ประสิทธิภาพการใช้งาน และความทนทานของระบบ
  - สิ่งเหล่านี้ไม่สามารถจำลองหรือสร้างขึ้นในสภาพแวดล้อมการทดสอบชนิดอื่น ๆ ที่กล่าวมาก่อนหน้านี้

Types of user testing

- การทดสอบอัลฟ่า (Alpha testing)
  - ผู้ใช้ซอฟต์แวร์ทำงานร่วมกับทีมพัฒนาเพื่อทดสอบซอฟต์แวร์ในที่ตั้งของผู้พัฒนาซอฟต์แวร์
- การทดสอบเบต้า (Beta testing)
  - มีการเปิดตัวซอฟต์แวร์เพื่อให้ผู้ใช้สามารถทดลองและแจ้งปัญหาที่พบกับนักพัฒนาระบบได้
- การทดสอบการยอมรับ
  - ลูกค้าทดสอบระบบ เพื่อตัดสินใจว่าระบบนั้นพร้อมที่จะได้รับการยอมรับ ทั้งจากนักพัฒนาระบบและใช้งานในสภาพแวดล้อมของลูกค้าหรือไม่

The acceptance testing process



Stages in the acceptance testing process

- กำหนดเกณฑ์การยอมรับ (Define acceptance criteria)
- วางแผนการทดสอบการยอมรับ (Plan acceptance testing)
- ได้รับการทดสอบการยอมรับ (Derive acceptance tests)
- เรียกใช้การทดสอบการยอมรับ (Run acceptance tests)
- เจรจาผลการทดสอบ (Negotiate test results)
- ปฏิเสธ / ยอมรับระบบ (Reject/accept system)

Agile methods and acceptance testing

- ในวิธีอไจล์ ผู้ใช้/ลูกค้าเป็นส่วนหนึ่งของทีมพัฒนาและมีหน้าที่รับผิดชอบในการตัดสินใจเกี่ยวกับการยอมรับระบบ
- การทดสอบถูกกำหนดโดยผู้ใช้/ลูกค้าและรวมเข้ากับการทดสอบอื่น ๆ โดยอัตโนมัติเมื่อมีการเปลี่ยนแปลง
- ไม่มีกระบวนการทดสอบการยอมรับที่แยกออกไปต่างหาก
- ปัญหาหลักคือ ผู้ใช้ที่ฝังตัวอยู่ในทีมพัฒนาไม่สามารถเป็นตัวแทนผู้มีส่วนได้เสียสำหรับทุกส่วนของระบบ

Key points

- สิ่งที่ test ทำคือสามารถแสดงเฉพาะข้อผิดพลาดในโปรแกรมได้ แต่ไม่สามารถแสดงให้เห็นว่าไม่มีข้อผิดพลาดใด ๆ หลงเหลืออยู่
- การพัฒนา testing เป็นความรับผิดชอบของทีมพัฒนาซอฟต์แวร์ ส่วนการทดสอบระบบก่อนที่จะเผยแพร่ให้กับลูกค้า ควรเป็นความรับผิดชอบของทีมที่แยกจากกันโดยอิสระ
- การพัฒนาการทดสอบประกอบด้วย
  - unit test จะทดสอบแต่ละชิ้นส่วน
  - component testing จะทดสอบกลุ่มของวัตถุ
  - system testing จะทดสอบระบบบางส่วนหรือแบบสมบูรณ์

- ควรลอง 'แบ่งส่วน' ซอฟต์แวร์ที่จะทดสอบ โดยใช้ประสบการณ์และแนวทางในการเลือกประเภทของ test case ที่มีประสิทธิภาพในการค้นพบข้อบกพร่องจากที่พบในระบบอื่น ๆ
- ถ้าเป็นไปได้ควรเขียนการทดสอบอัตโนมัติ ซึ่งการทดสอบจะฝังอยู่ในโปรแกรมที่สามารถเรียกใช้ได้ทุกครั้งที่มีการเปลี่ยนแปลงระบบ
- การทดสอบก่อนพัฒนา (Test-first development) เป็นแนวทางในการพัฒนาซึ่งจะมีการเขียน test ที่จะทดสอบก่อนที่จะเขียน code
- การทดสอบการยอมรับคือกระบวนการทดสอบของผู้ใช้ โดยมีวัตถุประสงค์เพื่อพิจารณาว่าซอฟต์แวร์มีดีพอที่จะติดตั้งและใช้งานในสภาพแวดล้อมการดำเนินงานของผู้ใช้ได้หรือไม่

คำถาม???