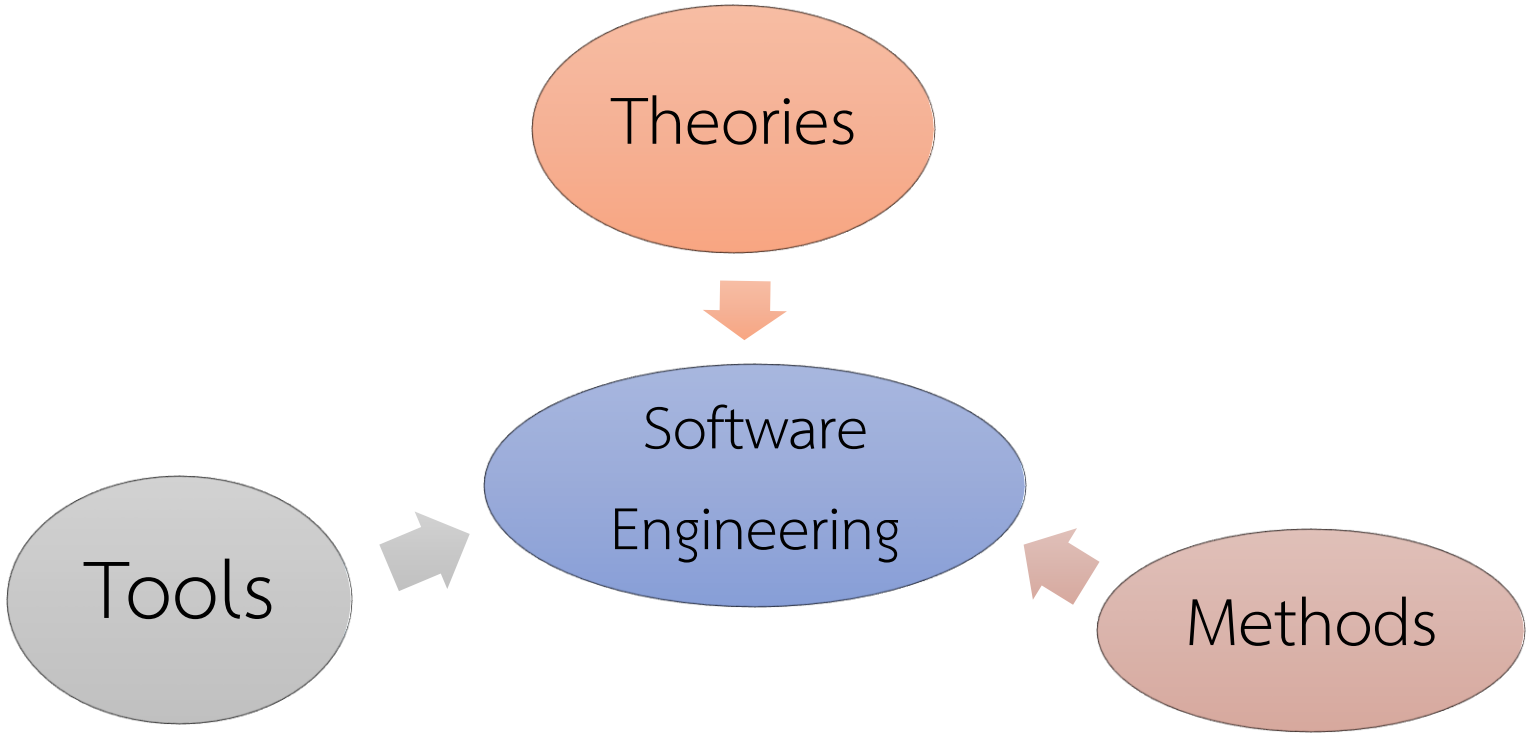
Software Processes

Week 2

ดัดแปลงจาก slides ของหนังสือ Software Engineering [1]

วิศวกรรมซอฟต์แวร์



- Computer Architecture
- Programming Language
- Programming paradigm
- .....

หัวข้อที่จะศึกษา

2.1 แบบจำลองการพัฒนาซอฟต์แวร์ (Software process models)

2.2 กิจกรรมในกระบวนการพัฒนา (Process activities)

2.3 การรับมือกับการเปลี่ยนแปลงในซอฟต์แวร์ (Coping with change)

2.4 การปรับปรุงกระบวนการ (Process improvement)

2.1 แบบจำลองการพัฒนาซอฟต์แวร์

Software process models

กระบวนการพัฒนาซอฟต์แวร์

- กิจกรรมต่างๆ ที่มีโครงสร้างเป็นระบบ ที่จำเป็นในการพัฒนาระบบซอฟต์แวร์
- กระบวนการพัฒนาซอฟต์แวร์ มีเยอะแยะมากมาย แต่มีส่วนที่เหมือนกัน ได้แก่
  - ข้อกำหนด (Specification) - การกำหนดสิ่งที่ระบบควรทำ
  - การออกแบบและการสร้าง (Design and Implementation) - การกำหนดองค์ประกอบและการสร้างซอฟต์แวร์
  - การตรวจสอบความถูกต้อง (Validation) - การตรวจสอบว่าซอฟต์แวร์ทำในสิ่งที่ลูกค้าต้องการ
  - วิวัฒนาการ (Evolution) – การปรับเปลี่ยนซอฟต์แวร์เพื่อตอบสนองความต้องการของลูกค้าที่เปลี่ยนแปลงไป
- Software process model เป็นตัวกำหนดนิยามกระบวนการผลิตซอฟต์แวร์ กล่าวถึงสิ่งต่อไปนี้
  - กิจกรรมที่ต้องทำในการพัฒนาซอฟต์แวร์ เช่น การกำหนดนิยามข้อมูล ออกแบบส่วนติดต่อผู้ใช้ เป็นต้น
  - ลำดับขั้นตอนในการพัฒนาซอฟต์แวร์

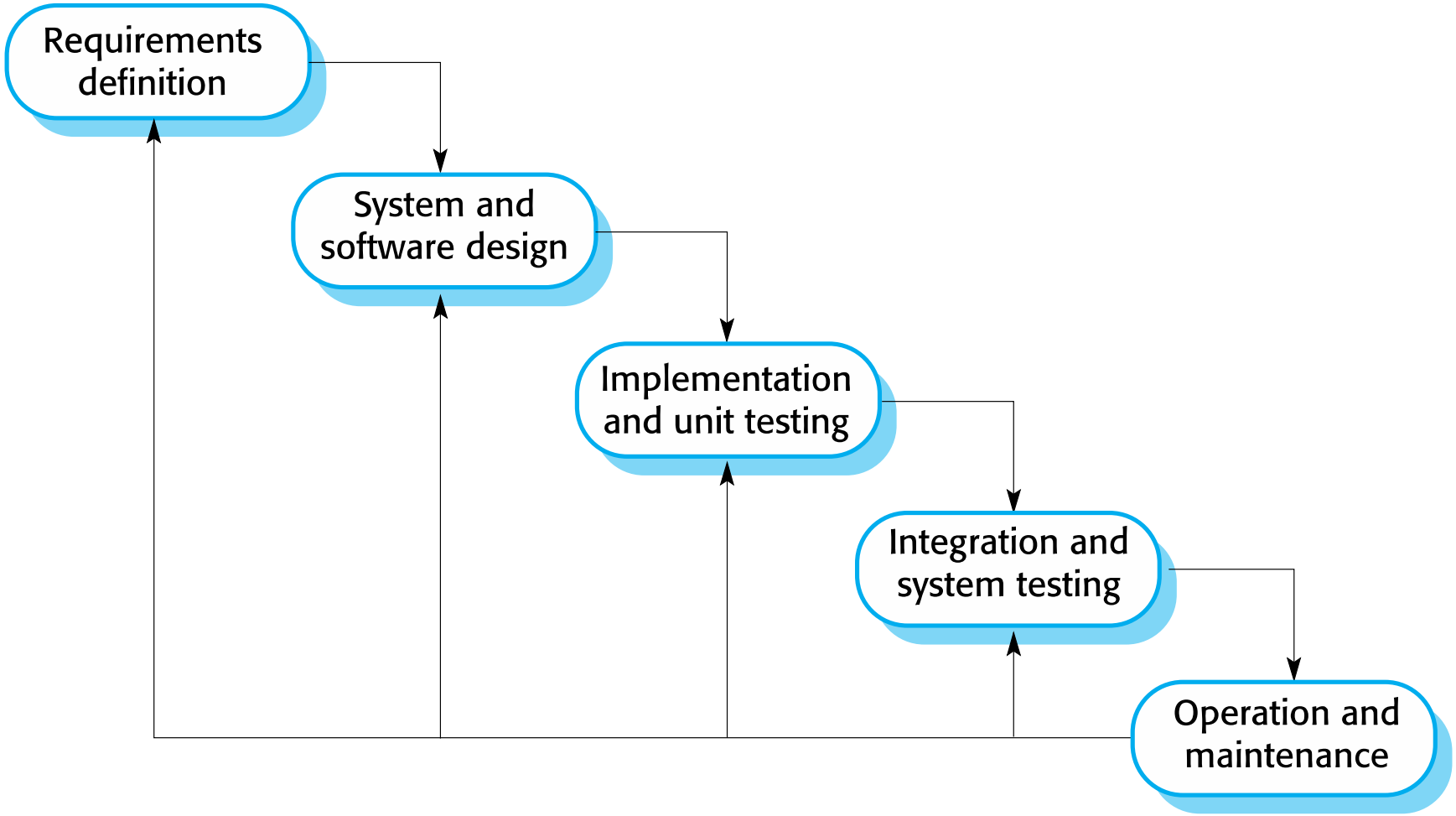
Software process descriptions

- รายละเอียดในกระบวนการพัฒนาซอฟต์แวร์
  - ผลิตภัณฑ์ (Products) - เป็นผลผลิตที่ได้จากกระบวนการพัฒนา ประกอบด้วย อะไรบ้าง?
  - บทบาท (Roles) – ระบุความรับผิดชอบของคนที่เกี่ยวข้องในกระบวนการ ประกอบด้วย ใครบ้าง?
  - เงื่อนไขก่อนและหลัง (Pre- and post-conditions) – เป็นข้อความที่ระบุข้อเท็จจริงทั้งก่อนและหลังการดำเนินกระบวนการหรือการสร้างผลิตภัณฑ์ได้สำเร็จ
- ในการพัฒนาซอฟต์แวร์ เราไม่สามารถบอกได้ชัดเจนว่า กระบวนการแบบใดถูกหรือผิด แต่...
  - คุ้มค่า
  - เวลาในการพัฒนา
  - การบำรุงรักษา
  - การบริการทรัพยากร เช่น programmer

Software process models

- The waterfall model
  - เป็นโมเดลแบบ Plan-driven – มีการแยกส่วน specification และ development อย่างชัดเจน
- Incremental development
  - อาจมีการทับซ้อนกันในส่วนของ Specification, development และ validation
  - เป็นได้ทั้งแบบ plan-driven หรือ agile
- Integration and configuration
  - ระบบที่ถูกสร้างจากระบบที่มีอยู่ (ซึ่งถูกออกแบบให้เป็น component ที่ configurable)
  - เป็นได้ทั้งแบบ plan-driven หรือ agile
- ในทางปฏิบัติ ระบบขนาดใหญ่ถูกพัฒนาขึ้นจากกระบวนการที่หลากหลายและอาจจะใช้ทุกแบบจำลองที่มีอยู่
  - ทรัพยากรที่มีจำกัดคือ คน การเลือกใช้โมเดลใดมักจะขึ้นอยู่กับความเชี่ยวชาญของคนเป็นปัจจัยหลัก

The waterfall model



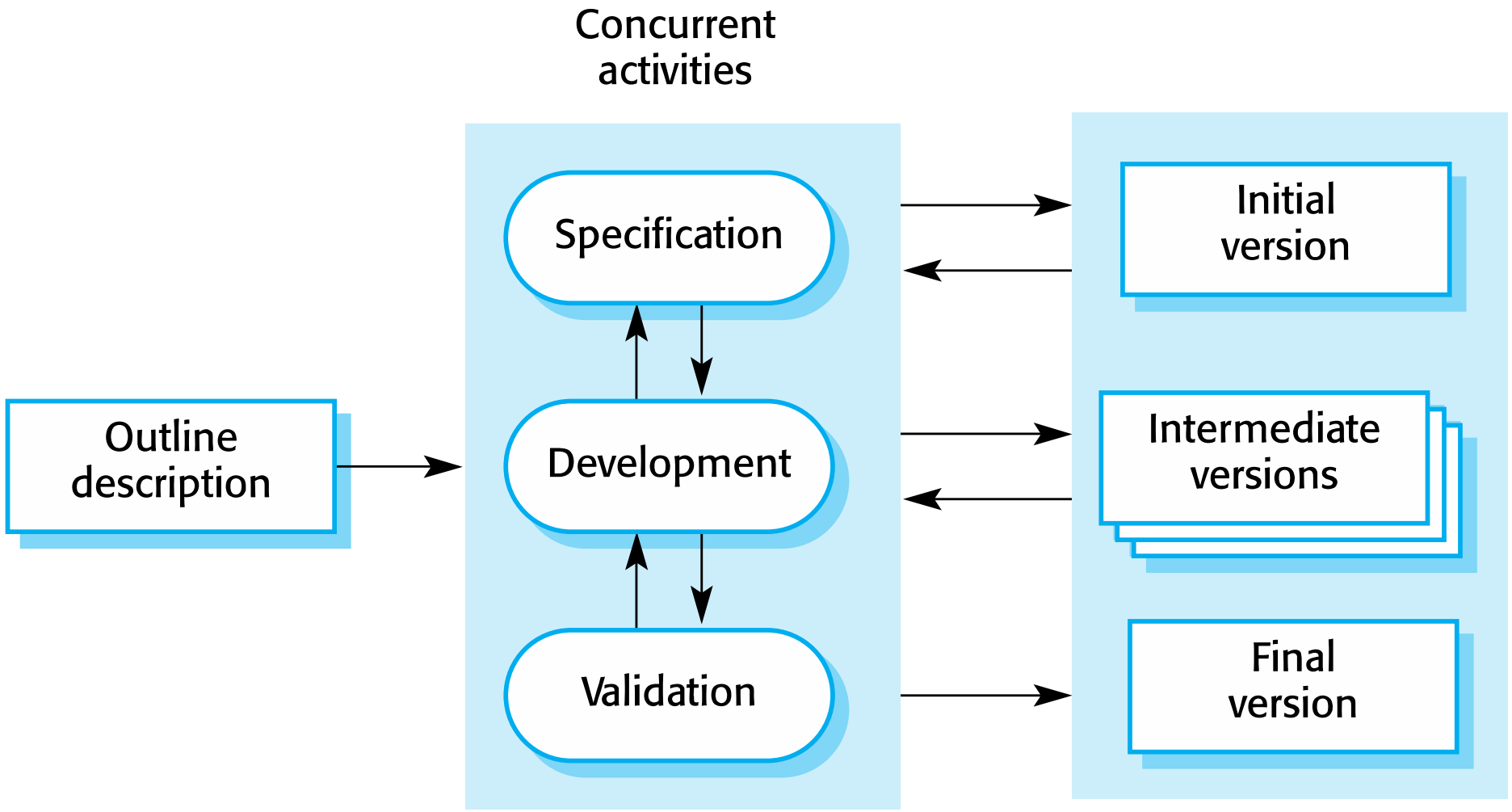
Waterfall model phases

- มีขั้นตอนที่แยกจากกันอย่างชัดเจน
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- ข้อจำกัดที่สำคัญของแบบจำลองน้ำตกคือ ความยากลำบากในการเปลี่ยนแปลงในขณะที่กระบวนการในแต่ละขั้นเริ่มดำเนินการไปแล้ว โดยหลักการแล้ว แต่ละ phase จะต้องเสร็จสมบูรณ์ก่อนจะก้าวสู่ phase ถัดไป

Waterfall model problems

- ตอบสนองต่อการเปลี่ยนแปลงความต้องการของลูกค้าได้ยาก
  - กระบวนการแบบน้ำตก จะใช้ได้ผลดี เมื่อรู้ความต้องการที่ชัดเจน
  - ระบบส่วนใหญ่ มักจะไม่มีความต้องการที่ชัดเจนและตายตัว ดังนั้นจึงเป็นไปไม่ได้ ที่จะไม่มีการเปลี่ยนแปลงความต้องการ ในขณะที่กำลังดำเนินกระบวนการในขั้นตอนต่างๆ
- กระบวนการแบบน้ำตก นิยมใช้ในโครงการขนาดใหญ่
  - อาจจะเป็นโครงการที่แยกส่วนย่อย เพื่อช่วยกันสร้างแล้วนำกลับมารวมกันในภายหลัง
  - อาจจะใช้แนวทางที่เรียกว่า plan-driven ในการขับเคลื่อนระบบ

Incremental development



Incremental development benefits

- ค่าใช้จ่ายในการรองรับการเปลี่ยนแปลงความต้องการของลูกค้าจะลดลง
  - ปริมาณของการวิเคราะห์และเอกสารที่จะต้องทำใหม่ในแต่ละขั้นตอน มีน้อยกว่าแบบจำลองน้ำตก
- สามารถที่จะรับข้อมูลป้อนกลับจากลูกค้าได้เร็วกว่าแบบจำลองน้ำตก
  - ลูกค้าสามารถแสดงความคิดเห็น จากซอฟต์แวร์ต้นแบบได้ทันที และสามารถรับรู้ถึงความคืบหน้าในการพัฒนาซอฟต์แวร์ของตนเอง
- สามารถส่งซอฟต์แวร์ในส่วนที่สำคัญไปให้ลูกค้าใช้งานก่อน
  - ลูกค้าได้รับประโยชน์จากเงินลงทุนได้เร็วกว่าแบบจำลองน้ำตก

Incremental development problems

- ไม่สามารถเห็นกระบวนการพัฒนาที่ชัดเจนได้
  - ดูเหมือนว่าต้องมีการส่งมอบงานบ่อย (ใน intermediate version) เพื่อให้เห็นถึงความคืบหน้าของงาน
  - การทำเอกสารที่สอดคล้องกับทุกรุ่นที่มีการเปลี่ยนแปลงทำได้ยากมาก และอาจจะไม่คุ้มทุน
- โครงสร้างของระบบ อาจจะแย่ลงเมื่อมีการเพิ่มเติมเนื้องานตามความต้องการมากขึ้น
  - การทำ refactoring ในขณะปรับปรุงซอฟต์แวร์เป็นจำนวนรุ่นย่อย ๆ ที่มากเกินไป จะทำให้เกิดความสิ้นเปลืองเป็นอย่างมาก
  - ถ้าทำเอกสารไม่ดี จะไม่สามารถติดตามการเปลี่ยนแปลงได้

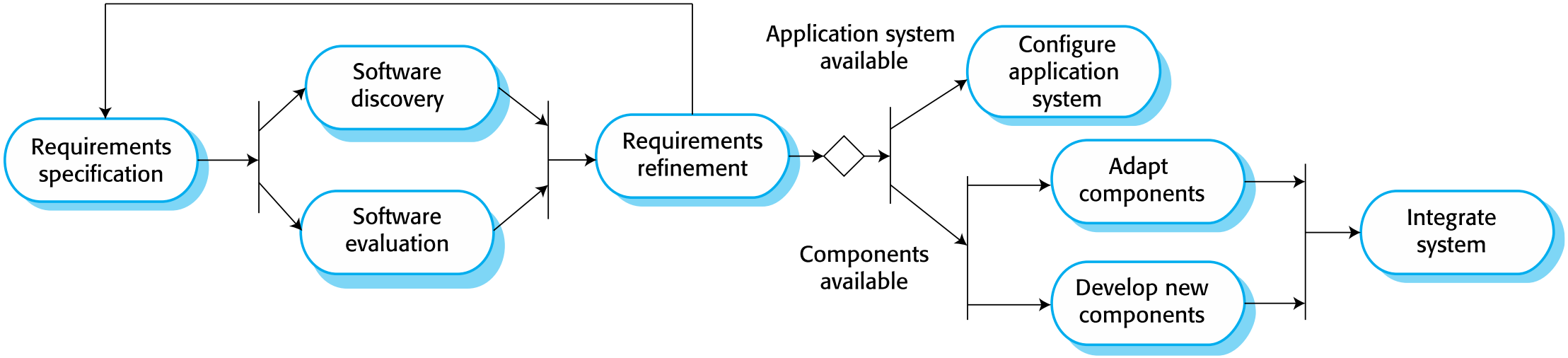
Integration and configuration

- กระบวนการนี้ จะอยู่บนพื้นฐาน software reuse โดยทั้งระบบจะเป็นการนำ software ที่มีอยู่แล้วมาทำการ config เพื่อให้เข้ากับความต้องการของลูกค้า
  - บางที่จะเรียกว่า COTS ย่อมาจาก Commercial-off-the-shelf
- ชิ้นส่วนของ software จะถูก configured เสียใหม่ เพื่อให้มี behaviour และ functionality ที่ตรงตาม requirement ของผู้ใช้
- ปัจจุบันถือว่า reuse เป็นวิธีการมาตรฐานอย่างหนึ่งในการพัฒนาซอฟต์แวร์
  - เราจะเรียนในเรื่องการเขียนซอฟต์แวร์ให้ใช้ได้ใหม่ (Software Reuse)

Types of reusable software

- Application แบบ Stand-alone (บางทีก็เรียก COTS) เป็นระบบที่นำซอฟต์แวร์สำเร็จ มาconfigured ใหม่ เพื่อให้เข้ากับสภาพแวดล้อมที่ลูกค้าต้องการ
- Collections ของ objects หรือชิ้นส่วนซอฟต์แวร์ ที่ถูกพัฒนาขึ้น เพื่อทำงานร่วมกับ component framework เช่น .NET หรือ J2EE
- Web services ที่ถูกพัฒนาขึ้นตาม service standards และสามารถถูกเรียกใช้จากระยะไกล ผ่านเว็บบราวเซอร์ หรือระบบอื่น ๆ

Reuse-oriented software engineering



Key process stages

- กำหนดความต้องการ (Requirement's specification)
- ค้นหาและประเมินซอฟต์แวร์ที่มีอยู่แล้ว (Software discovery and evaluation)
- ปรับปรุงความต้องการ (Requirement's refinement) ให้สอดคล้องกับซอฟต์แวร์ที่มีอยู่
- ปรับแต่งซอฟต์แวร์ (Application system configuration)
- ปรับแต่งและรวมชิ้นส่วนซอฟต์แวร์ (Component adaptation and integration)

Advantages and disadvantages

- ลดต้นทุนและความเสี่ยง เนื่องจากพัฒนาซอฟต์แวร์ขึ้นใหม่เป็นจำนวนน้อย
- ส่งมอบได้เร็ว
- อาจจะไม่ตรงกับความต้องการทั้งหมด/ที่แท้จริงของผู้ใช้
  - อาจจะต้องทำ refinement หรือ develop บางชิ้นส่วนของซอฟต์แวร์ใหม่
- ไม่สามารถควบคุม evolution หรือการ reused ของชิ้นส่วน
  - บางชิ้นส่วนที่ถูกแก้ไขโดยเจ้าของ อาจไม่ backward compatible กับรุ่นที่เรานำมาปรับใช้

2.2 กิจกรรมในกระบวนการพัฒนา

Process activities

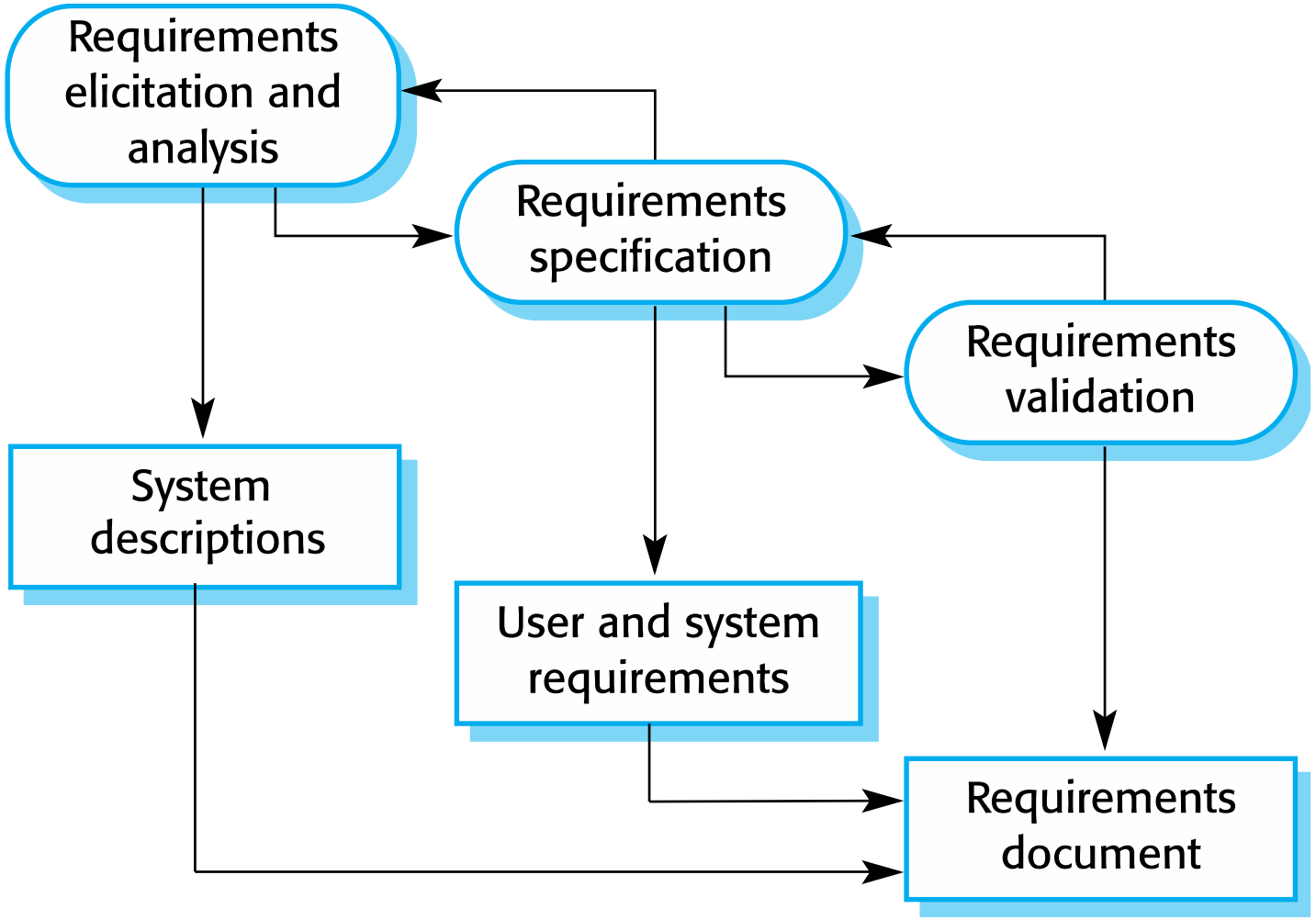
Process activities

- กระบวนการพัฒนาซอฟต์แวร์ เป็นกระบวนการที่เกี่ยวเนื่องกันของการทำงานเทคนิคหลาย ๆ ด้าน ที่ต้องอาศัยความร่วมมือและการจัดการที่ดี เพื่อเป้าหมายร่วมกันคือ ความสำเร็จของโครงการซอฟต์แวร์
  - กระบวนการประกอบด้วย specifying, designing, implementing และ testing
- กิจกรรมทั้งหมดในกระบวนการพื้นฐานสี่อย่าง (ได้แก่ specification, development, validation และ evolution) ของแต่ละ process model ต่างก็มีรายละเอียดที่และขั้นตอนที่แตกต่างกัน
  - เช่น ในแบบจำลองน้ำตกจะดำเนินการให้จบไปทีละขั้น แต่ในแบบจำลอง incremental นั้น จะทำสลับกันไป
  - ความสำคัญคือ เราต้องจัดตารางทรัพยากรให้เหมาะกับกระบวนการพัฒนาแต่ละชนิด

The requirements engineering process

กิจกรรม

ผลผลิต



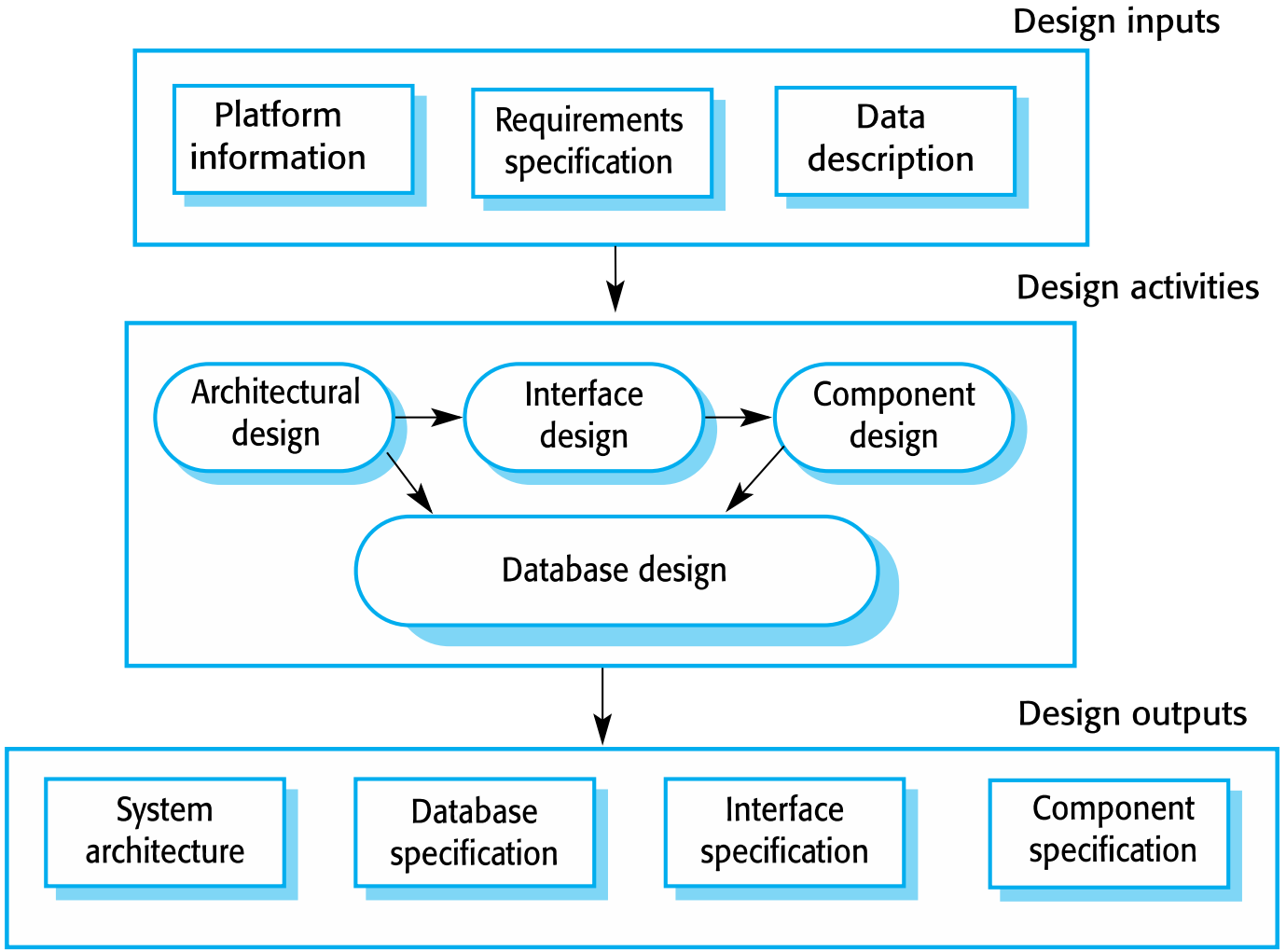
Software specification

- เป็นกระบวนการกำหนดสิ่งที่ต้องการ รวมถึงข้อจำกัดต่างๆ ในการดำเนินงานและพัฒนาระบบ
- Requirements engineering process
  - Requirements elicitation and analysis
    - สิ่งที่ผู้มีส่วนได้ส่วนเสียของระบบ (system stakeholders) ต้องการหรือมุ่งหวังจากระบบคืออะไร
  - Requirements specification
    - กำหนดรายละเอียดของ requirements
  - Requirements validation
    - ตรวจสอบความถูกต้องของ requirements (ตรวจสอบเอกสาร vs ความต้องการของ stakeholders)

Software design and implementation

- เป็น process ที่เปลี่ยน system specification ให้เป็น executable system.
- Software design
  - ออกแบบ software structure ที่ถูกต้องตรงตาม specification
- Implementation
  - เปลี่ยน software structure ให้เป็น executable program
- กิจกรรมในการ design และ implementation จะมีความเกี่ยวข้องใกล้ชิดกันมาก และอาจจะทำงานควบคู่กันไปได้

A general model of the design process



Design activities

- *Architectural design*
  - ใช้เมื่อต้องการออกแบบระบบในภาพรวม เพื่อให้ได้องค์ประกอบหลัก (principal components) ของระบบ (subsystems หรือ modules) เพื่อบอกความสัมพันธ์หรือการกระจัดกระจายขององค์ประกอบเหล่านั้น
- *Database design*
  - ใช้เมื่อต้องการออกแบบโครงสร้างข้อมูลของระบบ และบอกว่ามันจะไปอยู่ในฐานข้อมูลในลักษณะใด
- *Interface design*
  - ใช้เมื่อต้องการออกแบบการ interfaces ระหว่าง system components
- *Component selection and design*
  - ใช้เมื่อต้องการออกแบบระบบที่ใช้ชิ้นส่วนแบบ reuse ที่หาเจอ แต่ถ้ายังไม่มีใครทำไว้ ให้บอกว่ามันควรเป็นชิ้นส่วนที่ทำงานอย่างไร

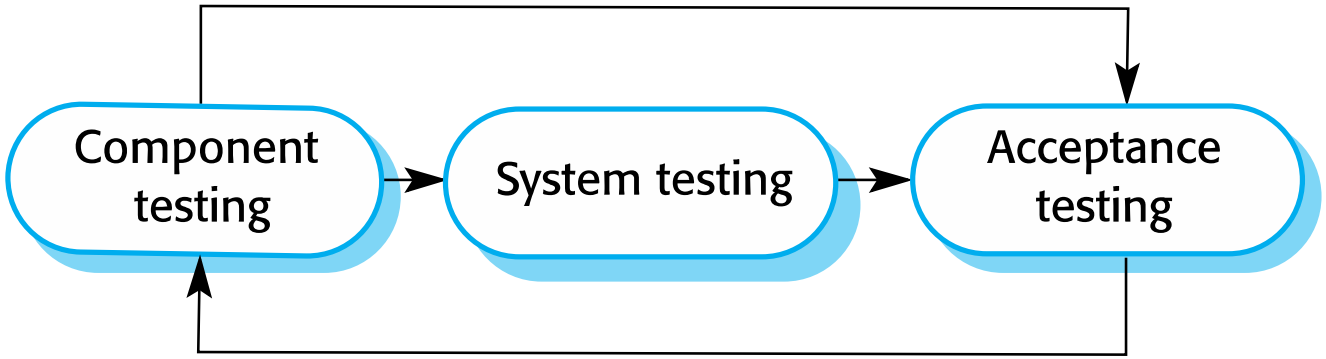
System implementation

- ซอฟต์แวร์ อาจจะเป็นได้ทั้งชนิดที่พัฒนาขึ้นใหม่ทั้งหมด หรือเป็นเพียงแค่การ configuring ซอฟต์แวร์ที่มีอยู่ ให้ตรงตามความต้องการของผู้ใช้
- การออกแบบ (design) และสร้าง (implementation) เป็นกิจกรรมที่ต้องทำสลับกัน และเป็นกิจกรรมที่ต้องทำมากที่สุดในโครงการซอฟต์แวร์
- Programming เป็นกิจกรรมส่วนบุคคล ไม่มี standard process
- Debugging เป็นกิจกรรมเกี่ยวกับการหาและแก้ไขข้อบกพร่อง

Software validation

- Verification (การยืนยัน) และ validation (การตรวจสอบความถูกต้อง) หรือ V & V ใช้เพื่อแสดงให้เห็นว่าระบบมีความสอดคล้องหรือเข้ากันได้กับ specification และตรงตาม requirements ของ system customer.
- ประกอบด้วยกระบวนการ
  - checking และ review
  - system testing
- System testing ประกอบด้วยการรันระบบด้วย test cases ที่ได้มาจาก specification
  - ทดสอบด้วยข้อมูลจริงเพื่อให้เห็นพฤติกรรมของระบบ
- ในกิจกรรม V & V จะใช้ testing เป็นหลัก

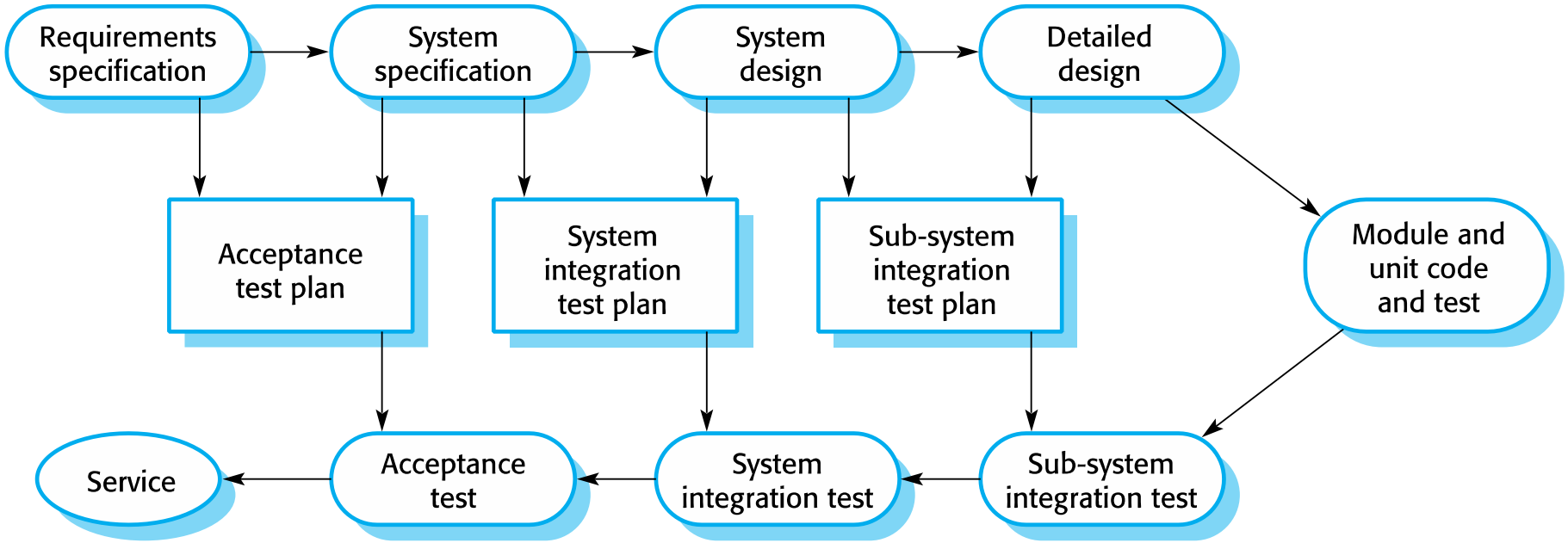
Stages of testing



Testing stages

- Component testing
  - แต่ละชิ้นส่วนถูก test โดยอิสระ
  - ชิ้นส่วนอาจจะหมายถึง functions หรือ objects
- System testing
  - เป็นการทดสอบระบบโดยรวมในคราวเดียว
  - ควรเน้นที่การทดสอบในส่วนที่ critical
- Customer testing
  - ทดสอบกับข้อมูลจริงของลูกค้า เพื่อตรวจสอบว่าระบบทำงานตรงตามความต้องการจริงของลูกค้าหรือไม่

Testing phases in a plan-driven software process (V-model)

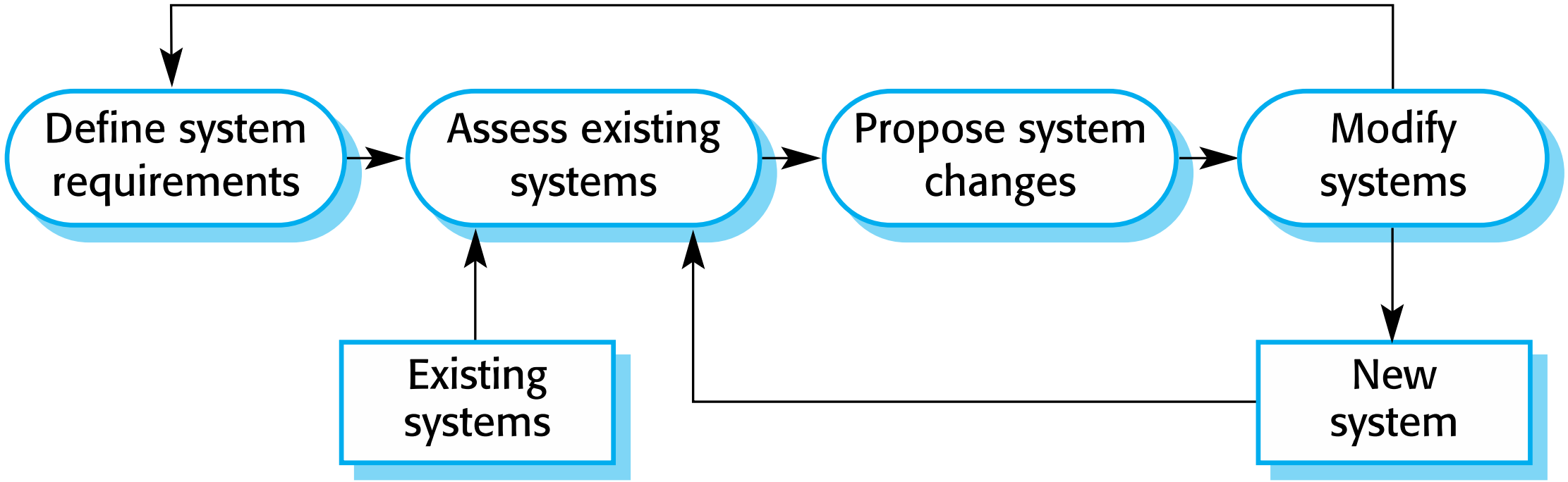


Software evolution

- ในระบบใดๆ ส่วนที่เป็น software ย่อมมีความยืดหยุ่นและสามารถเปลี่ยนแปลงได้
- ความต้องการซอฟต์แวร์อาจจะมีการเปลี่ยนแปลง
  - เช่น เกิดจากการเปลี่ยนแปลงพฤติการณ์ตามสภาพความเป็นจริงของธุรกิจ
  - ซอฟต์แวร์ที่สนับสนุนธุรกิจเหล่านั้น ก็ต้องรองรับการพัฒนาและเปลี่ยนแปลงอยู่เสมอ

- ให้นักศึกษาอธิบายความแตกต่างระหว่าง
  - การพัฒนา (development)
  - วิวัฒนาการ (evolution)
  - การบำรุงรักษา (maintenance)

System evolution



2.3 การรับมือกับการเปลี่ยนแปลงในซอฟต์แวร์

Coping with change

Coping with change

- ในโครงการซอฟต์แวร์ขนาดใหญ่ การเปลี่ยนแปลง เป็นสิ่งที่ไม่สามารถหลีกเลี่ยงได้ .
  - system requirements เปลี่ยนตาม business changes
  - เทคโนโลยีใหม่ๆ ช่วยให้เราสามารถพัฒนาซอฟต์แวร์ได้ง่ายขึ้น
  - platforms ที่ออกมาใหม่ๆ ต้องการ application ที่ต่างออกไปจากเดิม เช่น mobile phone
- ในการเปลี่ยนแปลง จะมีงานสองส่วนคือ
  - rework (เช่น re-analysing requirements)
  - implementing new functionality

Reducing the costs of rework

- ชิงลงมือก่อน โครงการซอฟต์แวร์สามารถทำบางอย่างให้เสร็จก่อนที่จะต้องมานั่งแก้งานในภายหลัง
  - เช่น ถ้าแน่ใจว่าลูกค้าจะต้องเพิ่ม features บางอย่างแน่นอน (แต่ดูเหมือนว่าจะยังนึกไม่ถึง) ให้รีบนำเสนอและพัฒนาเสียก่อนเลย
- ออกแบบให้เปลี่ยนแปลงได้ง่าย (หรือใช้ต้นทุนต่ำ)
  - กรณีนี้อาจจะทำได้ในกระบวนการพัฒนาแบบ incremental development
  - การเปลี่ยนแปลงอาจจะนำไปรวมกันไว้ใน increments ที่ยังไม่พัฒนาและพัฒนาในคราวเดียว

Coping with changing requirements

- ทำต้นแบบระบบ
  - ออกแบบและพัฒนาในส่วนสำคัญและพัฒนาได้เร็ว เพื่อตรวจสอบว่าตรงตามความต้องการของผู้ใช้หรือไม่
- ทยอยส่งมอบงาน
  - เหมาะกับการพัฒนาแบบ Incremental เพื่อรับคำแนะนำติชม และให้ลูกค้าได้ทดลองใช้ระบบในระยะเวลาหนึ่ง

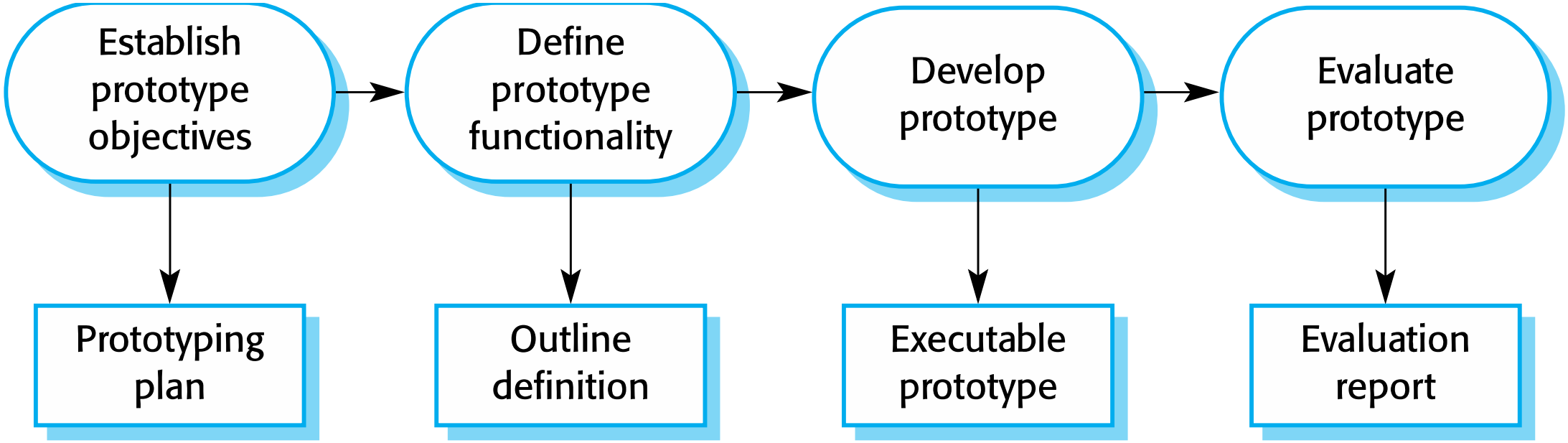
Software prototyping

- ต้นแบบ (prototype) ถือเป็น initial version ของ system
  - ใช้เพื่อสาธิต หรือนำเสนอแนวคิดของระบบ และให้ผู้ใช้ได้ทดลองใช้ นำไปสู่ทางเลือกในการพัฒนาระบบ
- ต้นแบบ สามารถนำมาใช้เมื่อใด?
  - ในขั้นตอน requirements เพื่อช่วยในการซักถามและเก็บ requirements ซึ่งสามารถนำไปใช้ได้ในกระบวนการ validation
  - ในขั้นตอน design เพื่อหาทางเลือกในการพัฒนาและออกแบบ User Interface
  - ในขั้นตอน testing เพื่อรัน back-to-back tests

Benefits of prototyping

- เพิ่มความเชื่อถือได้ของระบบ
- ตรงตามความต้องการของผู้ใช้
- เพิ่มคุณภาพในขั้นตอนการออกแบบ
- บำรุงรักษาง่าย
- ลดความยุ่งยากในการพัฒนา

The process of prototype development



Prototype development

- อาจพัฒนาด้วย tools หรือภาษาที่เหมาะกับการทำ rapid prototyping
- อาจจะตัด functionality บางอย่างออกไป
  - ต้นแบบ ควรมีเฉพาะสิ่งที่เข้าใจได้ยากหรืออาจจะนำไปสู่ความเข้าใจที่คลาดเคลื่อนระหว่าง user กับ developer
  - อะไรที่คุยกันเข้าใจง่ายๆ ไม่ต้องเสียเวลาทำต้นแบบ
  - ยังไม่ต้องเสียเวลากับการทำ Error checking หรือ recovery
  - เน้นที่ functional แทนที่จะทำ non-functional requirements
    - non-functional requirements เช่น reliability และ security

Throw-away prototypes

- เมื่อพัฒนาเสร็จ ควรเก็บ prototype ไว้ในที่ที่อยู่นอกพื้นที่การพัฒนา เนื่องจากมันอาจจะไม่สอดคล้องกับกระบวนการพัฒนาระบบ
  - เนื่องจากออกแบบอย่างลวกๆ (เพื่อทำความเข้าใจของทุกฝ่ายให้ตรงกัน) มันอาจจะไม่รองรับการพัฒนาเพื่อให้ตรงตาม requirement อื่น ๆ เช่น non-functional requirements
  - โดยปกติ การทำ prototypes มักจะไม่มีเอกสารต่างๆ ประกอบ
  - โดยทั่วไป prototype structure มักจะไปลดคุณภาพของกระบวนการพัฒนาซอฟต์แวร์
  - prototype โดยส่วนใหญ่มักจะไม่เข้ากับ quality standards เนื่องจากพัฒนาโดยภาษาหรือเครื่องมือที่เรียกว่า rapid prototype จึงมักจะขาดส่วนที่ควบคุมคุณภาพ

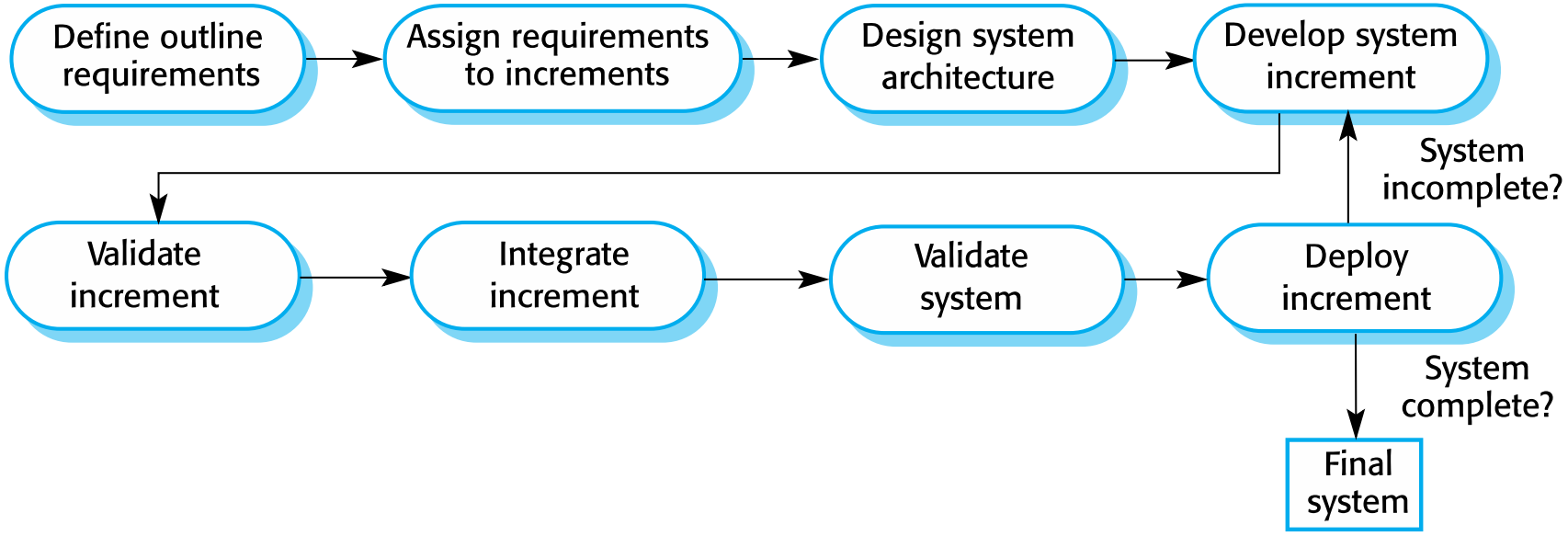
Incremental delivery

- แทนที่จะนำส่งระบบในรอบเดียว (single delivery) เราอาจจะแบ่งการพัฒนาและนำส่งออกเป็นหลาย ๆ รุ่น
  - แต่ละรุ่นอาจจะนำส่งตาม required functionality ที่แตกต่างกันไป
- ถ้ามีการนำส่งชนิดหลายรุ่น ให้ส่งซอฟต์แวร์ที่ตรงตาม USER REQUIREMENTS ก่อนเสมอ
  - ให้จัดลำดับความสำคัญให้ดี จัดตามความต้องการของผู้ใช้ ไม่ใช่จัดตามความชอบหรือถนัดของผู้พัฒนา
- เมื่อเริ่มพัฒนาในส่วน increment เราก็สามารถนำ requirement อื่น ๆ มาเริ่มพัฒนาได้

Incremental development and delivery

- Incremental development
  - พัฒนาระบบใน increment นั้นและทำการ evaluate แต่ละ increment ให้เสร็จก่อนที่จะขยับไปทำincrement ถัดไป
  - เป็นวิธีการปกติที่ใช้ใน agile methods (จะเรียนในหัวข้อที่ 3)
  - การทำ evaluation จะทำผ่าน user/customer proxy
- Incremental delivery
  - ส่งมอบ (deploy) increment ที่จะใช้โดย end-users
  - เป็นการ evaluation ซอฟต์แวร์ที่ตรงกับความเป็นจริงมากที่สุด;
  - สร้างระบบจำลองหรือทดแทนได้ยาก เนื่องจากแต่ละ increments จะมีความสามารถที่น้อยมาก เมื่อเทียบกับซอฟต์แวร์ที่สมบูรณ์

Incremental delivery



Incremental delivery advantages

- ส่งมอบสิ่งที่ลูกค้าต้องการมากที่สุดให้ก่อน ดังนั้นลูกค้าสามารถใช้งานได้ก่อน และได้ผลตอบแทนก่อน
- increments แรกๆ จะเป็นเสมือน prototype ให้กับลูกค้า ที่จะประเมินความต้องการ เพื่อที่จะนำไปสู่การพัฒนา increments ถัดไป
- มีความเสี่ยงน้อย ที่จะล้มเหลวทั้งโครงการ
- ส่วนที่ตรงกับ requirement มากที่สุด จะถูกนำไปใช้ก่อนและถูกทดสอบมากที่สุด

Incremental delivery problems

- ส่วนใหญ่แล้ว ความต้องการหลักของลูกค้า มักจะเป็นระบบพื้นฐานทีกระจัดกระจายไปตามส่วนต่างๆ ของทั้งระบบ
  - requirements ที่ชัดเจน มักจะยังไม่ออกมา จนกว่าจะถึงรอบการพัฒนา increment
  - ยากที่จะรู้ถึงความต้องการพื้นฐานที่แท้จริง ที่จำเป็นสำหรับทุก ๆ increments
- ปัญหาที่สำคัญคือ specification ใหม่ๆ มักจะเกิดคู่ขนานไปกับการพัฒนาทุก ๆ increment
  - ทำอย่างไร พัฒนาเท่าไร ก็ไม่จบสักที...
  - แต่ไม่ต้องกังวล เพราะโครงการซอฟต์แวร์ มักจะมี complete system specification เป็นส่วนประกอบของสัญญาจ้างพัฒนาซอฟต์แวร์ เราสามารถใช้สิ่งนั้นเป็นกรอบและข้อยุติในโครงการซอฟต์แวร์ได้

2.4 การปรับปรุงกระบวนการ

Process improvement

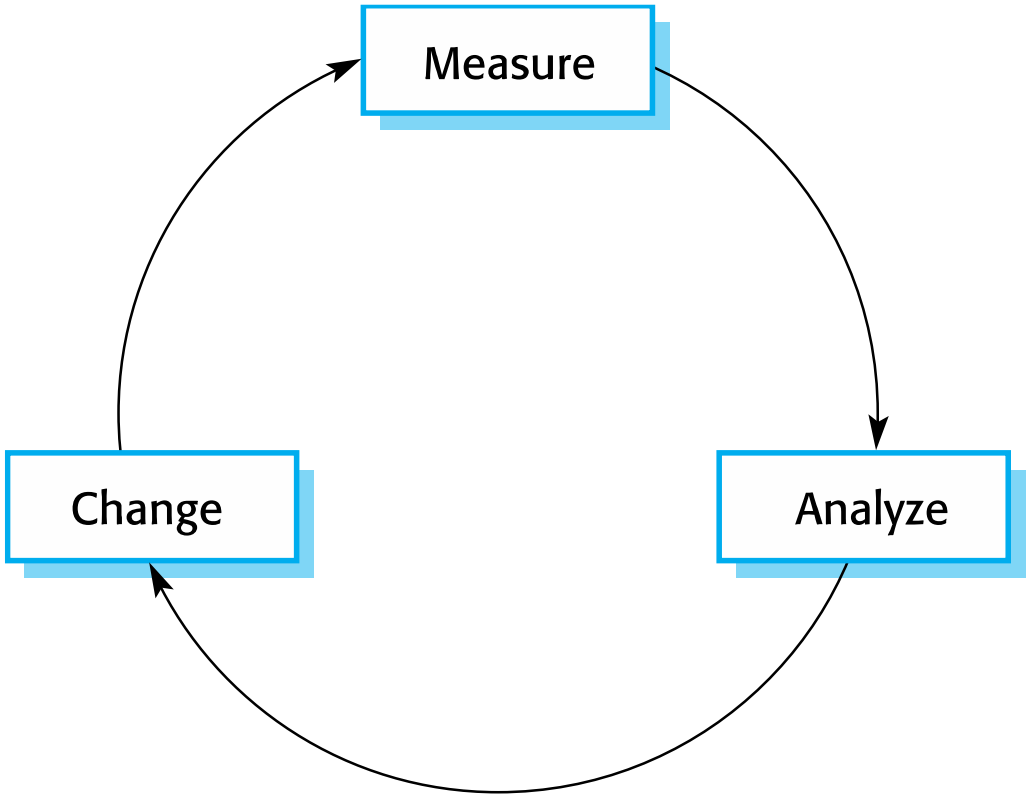
Process improvement

- บริษัทซอฟต์แวร์ส่วนใหญ่ นำ software process improvement มาเป็นแนวทางในการเพิ่มคุณภาพให้กับซอฟต์แวร์ ลดต้นทุนหรือแม้กระทั่งเพิ่มความเร็วในกระบวนการพัฒนาซอฟต์แวร์
- Process improvement หมายถึง การทำความเข้าใจกระบวนการที่มีอยู่เดิมแล้วทำการเปลี่ยนกระบวนการเหล่านั้น เพื่อเพิ่มคุณภาพผลผลิต และ/หรือลดต้นทุนรวมทั้งระยะเวลาที่ใช้ในการพัฒนา

Approaches to improvement

- กระบวนการกำหนดวุฒิภาวะ (process maturity approach)
  - มุ่งเน้นการปรับปรุงกระบวนการและการจัดการโครงการและแนะนำการปฏิบัติด้านวิศวกรรมซอฟต์แวร์ที่ดีมาใช้
  - ระดับของ process maturity จะสะท้อนถึงเทคนิคและการจัดการที่ดีในองค์กร
- The agile approach
  - มุ่งเน้นไปที่การพัฒนาซ้ำ (iterative development) และการลดค่าโสหุ้ย (overheads) ในกระบวนการซอฟต์แวร์
  - ลักษณะเบื้องต้นของวิธีการแบบ agile คือการส่งมอบ functionality และมีการตอบสนองอย่างรวดเร็วต่อการเปลี่ยนแปลงความต้องการของลูกค้า

The process improvement cycle



Process improvement activities

- *Process measurement*
  - วัดคุณลักษณะของ process หรือ product ของซอฟต์แวร์อย่างน้อยหนึ่งรายการ
  - การวัดเหล่านี้เป็นพื้นฐานที่ช่วยให้ตัดสินใจว่าการปรับปรุงกระบวนการทำงานได้ดีหรือไม่
- *Process analysis*
  - มีการประเมินกระบวนการปัจจุบัน ทำให้สามารถระบุจุดอ่อนของกระบวนการและปัญหาคอขวดที่มี
  - สิ่งที่จะได้คือ Process models (บางครั้งเรียกว่าแผนที่กระบวนการ process maps)
- *Process change*
  - ทำการเปลี่ยนแปลงกระบวนการ เพื่อแก้ไขจุดอ่อนของกระบวนการที่ระบุไว้
  - วนรอบกลับไปทำการรวบรวมข้อมูลเกี่ยวกับประสิทธิภาพของการเปลี่ยนแปลง

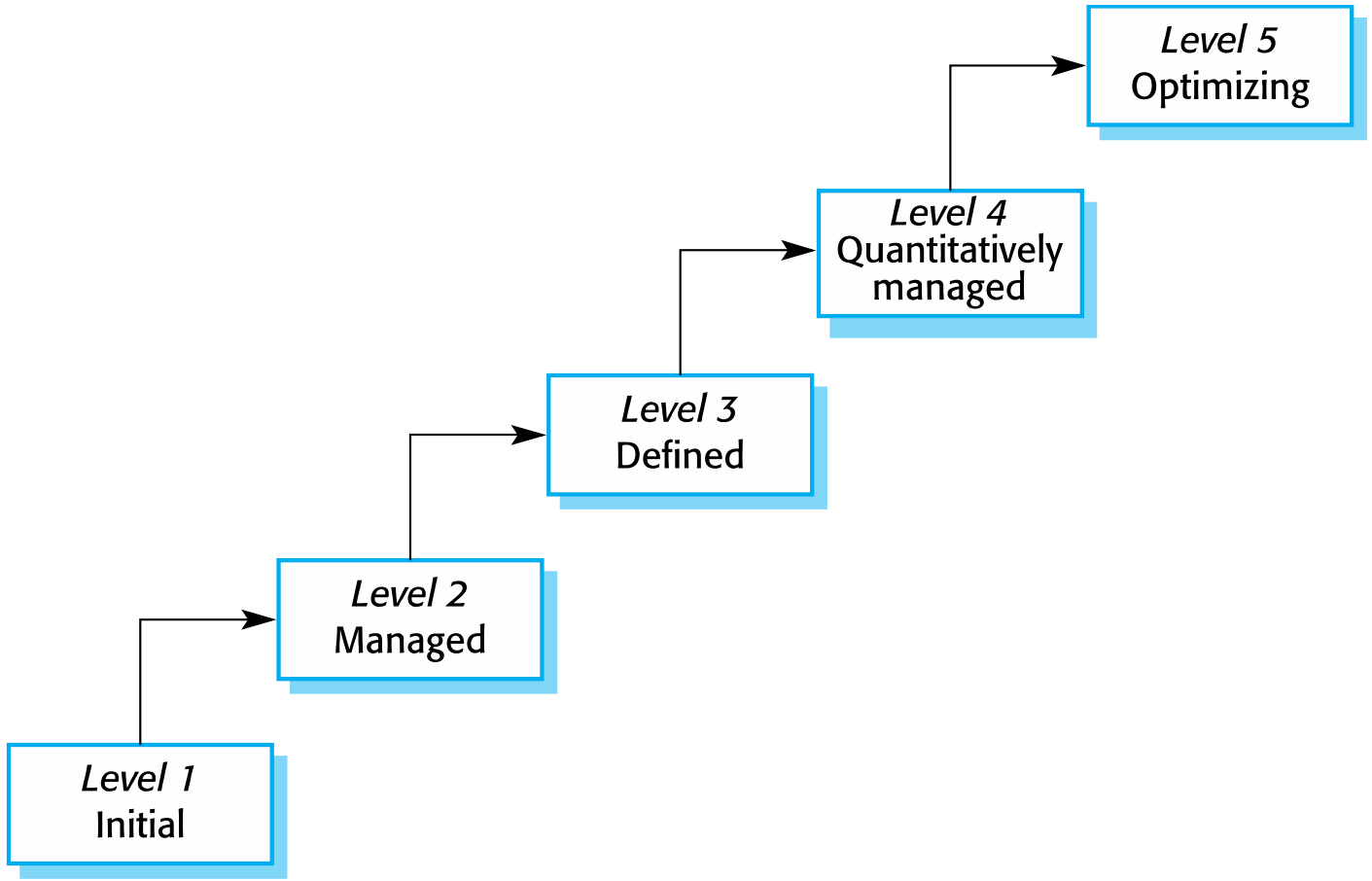
Process measurement

- ควรเก็บรวบรวมข้อมูลคุณภาพของกระบวนการให้มากเท่าที่จะทำได้
  - อย่างไรก็ตาม ในกรณีที่องค์กรไม่ได้กำหนดมาตรฐานกระบวนการไว้อย่างชัดเจน จะเป็นสิ่งที่ยากมาก เนื่องจากเราไม่ทราบว่าต้องวัดอะไร
  - อาจต้องมีการกำหนดกระบวนการก่อน แล้วค่อยวัดสิ่งที่ต้องการ
- ควรใช้การวัดกระบวนการเพื่อประเมินการปรับปรุงกระบวนการ
  - แต่ไม่ได้หมายความว่าเราต้องทำการวัดเพื่อกระตุ้นให้เกิดการปรับปรุงกระบวนการ
  - การกระตุ้นให้เกิดการปรับปรุงกระบวนการ ควรเป็นวัตถุประสงค์ขององค์กร

Process metrics

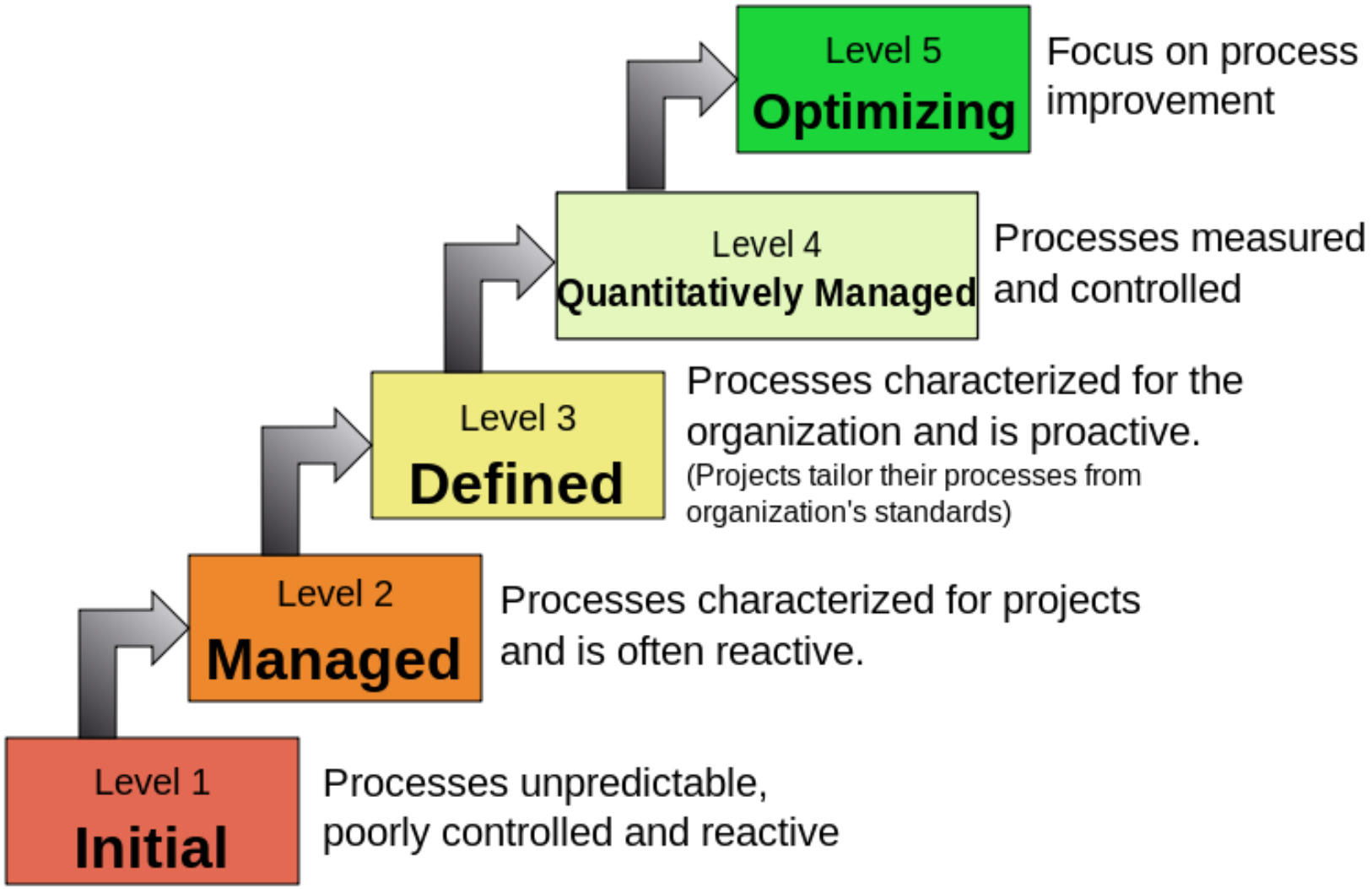
- เวลาที่ต้องใช้ในการดำเนินกิจกรรมของกระบวนการ
  - เช่น เวลาตามปฏิทินที่ต้องใช้จนกว่ากระบวนการจะแล้วเสร็จ
- ทรัพยากรที่ต้องใช้ในการดำเนินกิจกรรมของกระบวนการ
  - เช่น ชั่วโมงทำงานของคนงานทั้งหมด
- จำนวนเหตุการณ์พิเศษ
  - เช่นจำนวนชิ้นงานที่เสียหาย defect จากการผลิต

Capability maturity levels



http://www.tutorialspoint.com/cmmi/cmmi-maturity-levels.htm

Characteristics of the Maturity levels



https://www.wikiwand.com/en/Capability_Maturity_Model_Integration

The SEI capability maturity model

- Initial
  - ไม่มีการควบคุม
- Repeatable
  - มีการกำหนดและใช้งาน Product management procedures
- Defined
  - มีการกำหนดและใช้งาน Process management procedures and strategies
- Managed
  - มีการกำหนดและใช้งาน Quality management
- Optimising
  - มีการกำหนดและใช้งาน Process improvement strategies

Key points

- Software processes คือกิจกรรมทั้งหมดที่เกี่ยวข้องในการผลิตซอฟต์แวร์
- Software process models เป็นนิยาม (abstract) ของกระบวนการเหล่านี้
- General process models describe the organization of software processes.
- process models ทั่วไปอธิบายถึงกระบวนการจัดกระบวนการซอฟต์แวร์
  - เช่น waterfall model, incremental development, และ reuse-oriented development.
- Requirements engineering เป็นกระบวนการกำหนด software specification.
- Design และ implementation เป็นกระบวนการเปลี่ยนข้อกำหนดซอฟต์แวร์ให้กลายเป็นซอฟต์แวร์ที่ปฏิบัติการได้

Key points

- การตรวจสอบซอฟต์แวร์ (Software validation) เป็นกระบวนการตรวจสอบว่าระบบสอดคล้องกับข้อกำหนดและตรงกับความต้องการที่แท้จริงของผู้ใช้ระบบ
- Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.
- วิวัฒนาการของซอฟต์แวร์ (Software evolution) เกิดขึ้นเมื่อเราเปลี่ยนระบบซอฟต์แวร์ที่มีอยู่ ให้เป็นไปตามข้อกำหนดใหม่ ซอฟต์แวร์ต้องมีวิวัฒนาการเพื่อให้มีประโยชน์อยู่เสมอ
- Processes should include activities such as prototyping and incremental delivery to cope with change.
- กระบวนการ (Processes) หมายความรวมถึงกิจกรรมต่าง ๆ ทั้งหมดที่เกี่ยวข้อง เช่น การสร้างต้นแบบและการจัดส่ง incremental เพื่อรับมือกับการเปลี่ยนแปลง

Key points

- กระบวนการที่ใช้พัฒนาซอฟท์แวร์ อาจออกแบบสำหรับการพัฒนาและการส่งมอบซ้ำหลายรอบ (iterative development and delivery) เพื่อให้สามารถรองรับการเปลี่ยนแปลงได้โดยไม่กระทบกับระบบโดยรวม
- แนวทางหลักในการปรับปรุงกระบวนการคือ แนวทาง agile เพื่อลดต้นทุนโสหุ้ย และแนวทาง maturity-based ซึ่งทั้งสองแนวทางมี process management ที่ดี รวมถึงใช้หลักกี่วิศวกรรมซอฟต์แวร์ที่ดีด้วย
- SEI process maturity framework กำหนด maturity levels ที่จำเป็นต่อหลักวิศวกรรมซอฟต์แวร์ที่ดี (good software engineering practice)

อ้างอิง

[1] Ian Sommervielle, “Software Engineering”, 10th edition, Addison Wesley, 2015

Question?