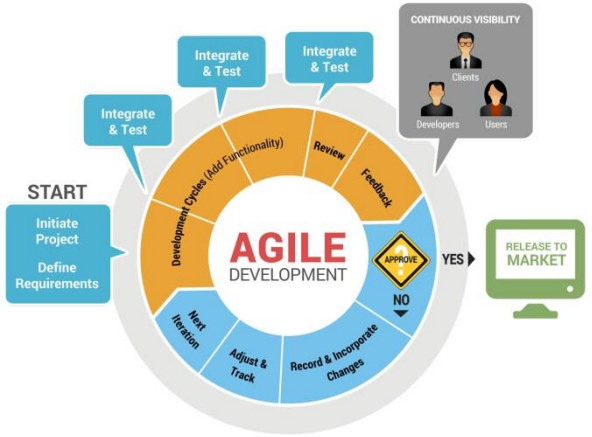
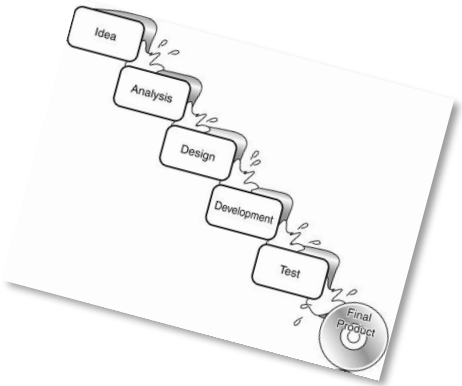


บทที่ 2
กระบวนการพัฒนาซอฟต์แวร์ (Software process)

วิชา วิศวกรรมซอฟต์แวร์ (04-06-322)



วัตถุประสงค์การเรียนรู้

- เพื่อให้ผู้เรียนมีความรู้ความเข้าใจเกี่ยวกับแนวคิดกระบวนการผลิตซอฟต์แวร์ และแบบจำลองกระบวนการผลิตซอฟต์แวร์
- เพื่อให้ผู้เรียนมีความรู้ความเข้าใจเกี่ยวกับกลุ่มของกิจกรรมที่เกี่ยวข้องกันที่เกิดขึ้นในการผลิตซอฟต์แวร์

หัวข้อ (Agenda)

- บทนำ (Overview)
- กระบวนการพัฒนาซอฟต์แวร์
  - กิจกรรมในกระบวนการ (Process Activities)
- แบบจำลองกระบวนการพัฒนาซอฟต์แวร์
- แนวทางการประยุกต์ใช้กระบวนการพัฒนาซอฟต์แวร์
- สรุป (Summary)

บทนำ (Overview)

กลุ่มของขั้นตอนการทำงาน

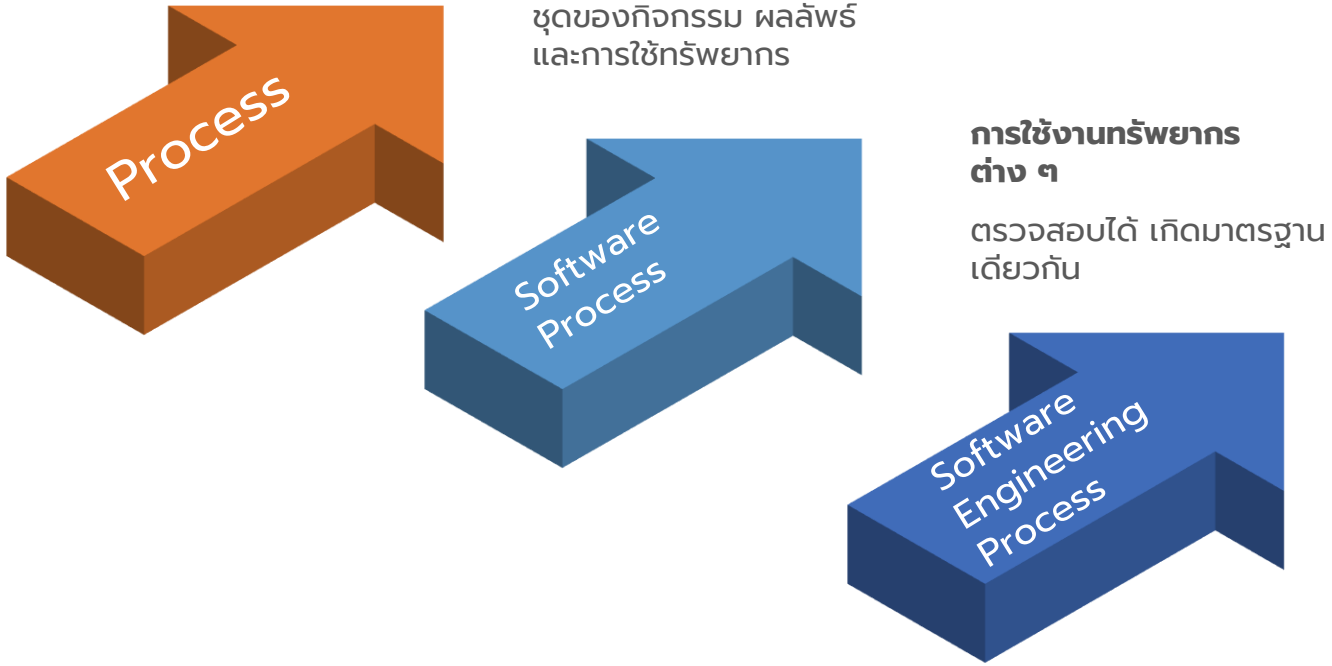
กิจกรรม ข้อจำกัด และ
ทรัพยากร

วิธีการปฏิบัติ

ชุดของกิจกรรม ผลลัพธ์
และการใช้ทรัพยากร

**การใช้งานทรัพยากร
ต่าง ๆ**

ตรวจสอบได้ เกิดมาตรฐาน
เดียวกัน



กระบวนการพัฒนาซอฟต์แวร์

กระบวนการ (Process) เป็นกลุ่มของขั้นตอนการทำงาน ประกอบด้วย กิจกรรม ข้อจำกัด และทรัพยากรที่ใช้ ผลิตผลลัพธ์

- ระบุกิจกรรมให้ชัดเจน ใช้ทรัพยากรภายใต้ข้อจำกัดต่าง ๆ
- ประกอบขึ้นจากกระบวนการย่อยอื่น ที่สัมพันธ์กัน
- ทุกกิจกรรมมีเงื่อนไข ในการเริ่มต้นและสิ้นสุด
- ทุกกิจกรรมมีเป้าหมายและหลักการที่ชัดเจน
- เงื่อนไขสามารถนำมาใช้ควบคุมการดำเนินกิจกรรม ทรัพยากร หรือผลิตภัณฑ์

Software product และ Software process

กระบวนการพัฒนาซอฟต์แวร์ (ต่อ)

"กรอบการดำเนินกิจกรรมในการสร้างซอฟต์แวร์ที่มีคุณภาพ"

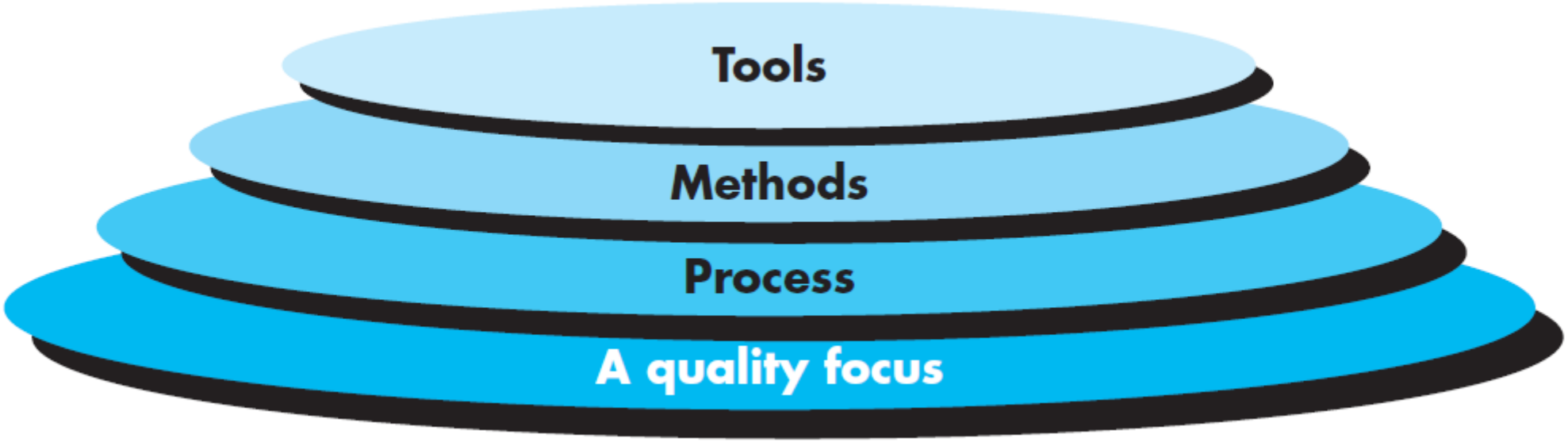
[Pressman, 2010]

"กลุ่มของกิจกรรมและผลลัพธ์ของแต่ละกิจกรรมเพื่อการผลิตซอฟต์แวร์ โดยมีกิจกรรมพื้นฐาน 4 กิจกรรม คือ Software specification, Software design and implementation, Software validation และ Software evolution"

[Sommerville, 2011]

"กลุ่มของกิจกรรมที่เกี่ยวเนื่องกันในการผลิตซอฟต์แวร์ให้ได้คุณภาพ โดยมีการกำหนดลำดับขั้นตอนไว้อย่างชัดเจนและสอดคล้องกัน ซึ่งทำให้เกิดประโยชน์ในการนำไปใช้งานต่อ"

วิศวกรรมซอฟต์แวร์เลเยอร์ (Software Engineering Layers)



- วิศวกรรมซอฟต์แวร์เลเยอร์ ประกอบด้วย เครื่องมือ (Tools) วิธีการ (Mehods) กระบวนการ (Process) และคุณภาพ (Quality)

ที่มา: Roger S. Pressman and Bruce R. Maxim, Software Engineering: A Practitioner's Approach, Eighth Edition, 2015.

กระบวนการพัฒนาซอฟต์แวร์ (ต่อ)

การกำหนดคุณลักษณะของซอฟต์แวร์

การออกแบบและนำเสนอซอฟต์แวร์

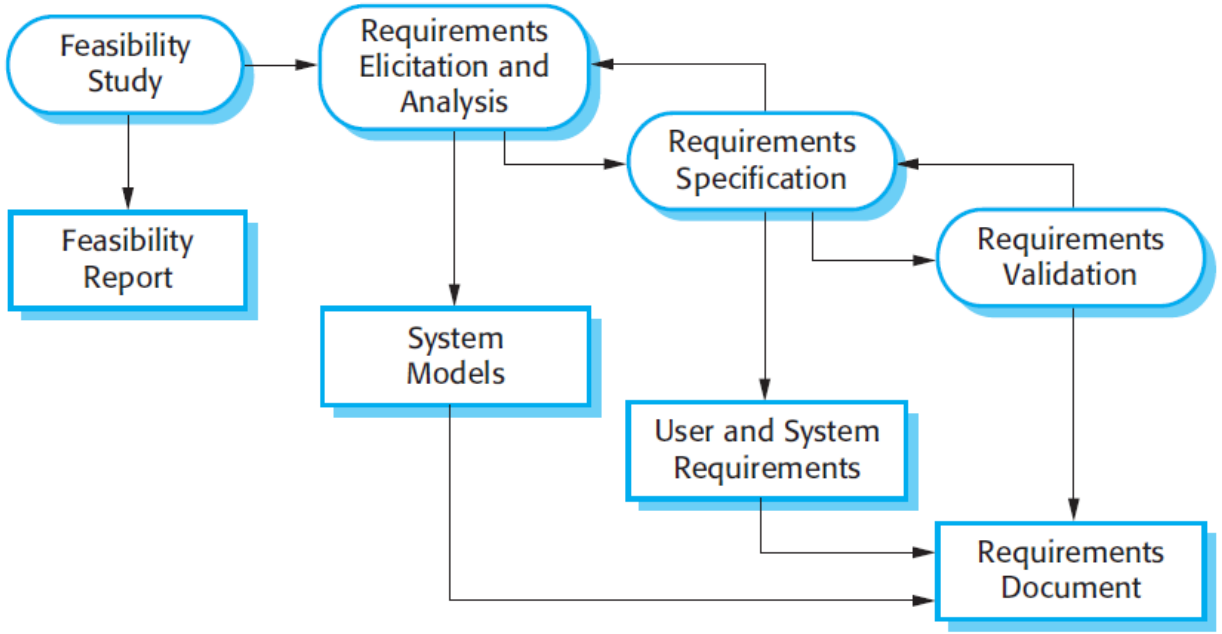
การตรวจสอบซอฟต์แวร์

การปรับปรุงและบำรุงรักษาซอฟต์แวร์

กิจกรรมในกระบวนการ (Process Activities)

การกำหนดคุณลักษณะของซอฟต์แวร์ (Software Specification)

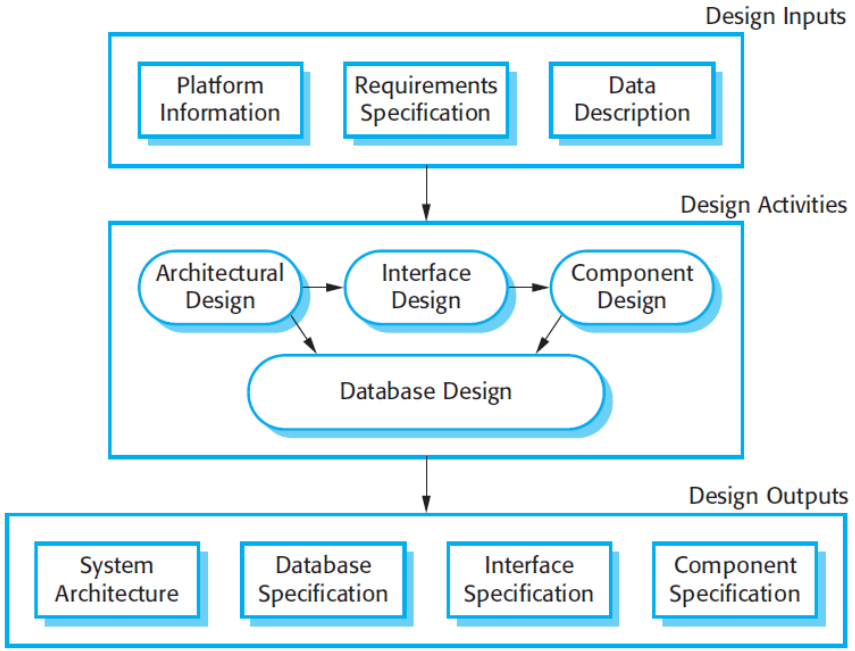
- การกำหนดคุณลักษณะของซอฟต์แวร์ หรือ วิศวกรรมความต้องการ (Requirements engineering) เป็นกระบวนสำหรับการทำความเข้าใจ กำหนดและนิยามเกี่ยวกับ
  - บริการใดที่จำเป็นสำหรับระบบ และ จำเป็นต้องกำหนดเป็นเงื่อนไขหรือข้อจำกัดในการดำเนินการและการพัฒนาระบบ
- วิศวกรรมความต้องการ



กิจกรรมในกระบวนการ (Cont.)

การออกแบบและนำเสนอซอฟต์แวร์ (Software Design and Implementation)

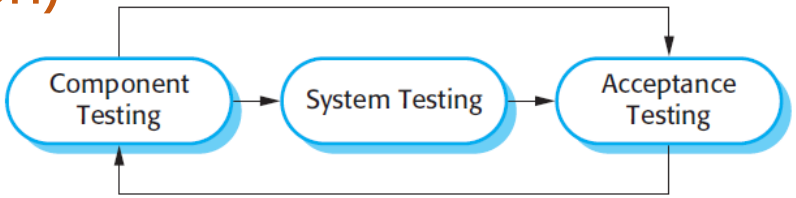
- การแปลงข้อกำหนดของระบบไปเป็นระบบที่สามารถประมวลผลหรือดำเนินการได้
  - กิจกรรมนี้ส่วนใหญ่จะเกี่ยวข้องกับกระบวนการของการออกแบบและเขียนโปรแกรม
  - วิธีการเพิ่มเติม (Incremental approach) เกี่ยวข้องกับการปรับแต่งข้อกำหนดของซอฟต์แวร์
- การออกแบบซอฟต์แวร์ (Software Design)
  - รายละเอียดโครงสร้างของซอฟต์แวร์ที่จะถูกนำเสนอ ดังนี้
    - แบบจำลองเชิงข้อมูลและเชิงโครงสร้างที่จะใช้ในระบบ
    - ส่วนต่อประสานระหว่างองค์ประกอบ (Component) ต่าง ๆ ของระบบ
    - อัลกอริทึมที่ใช้ในการพัฒนา

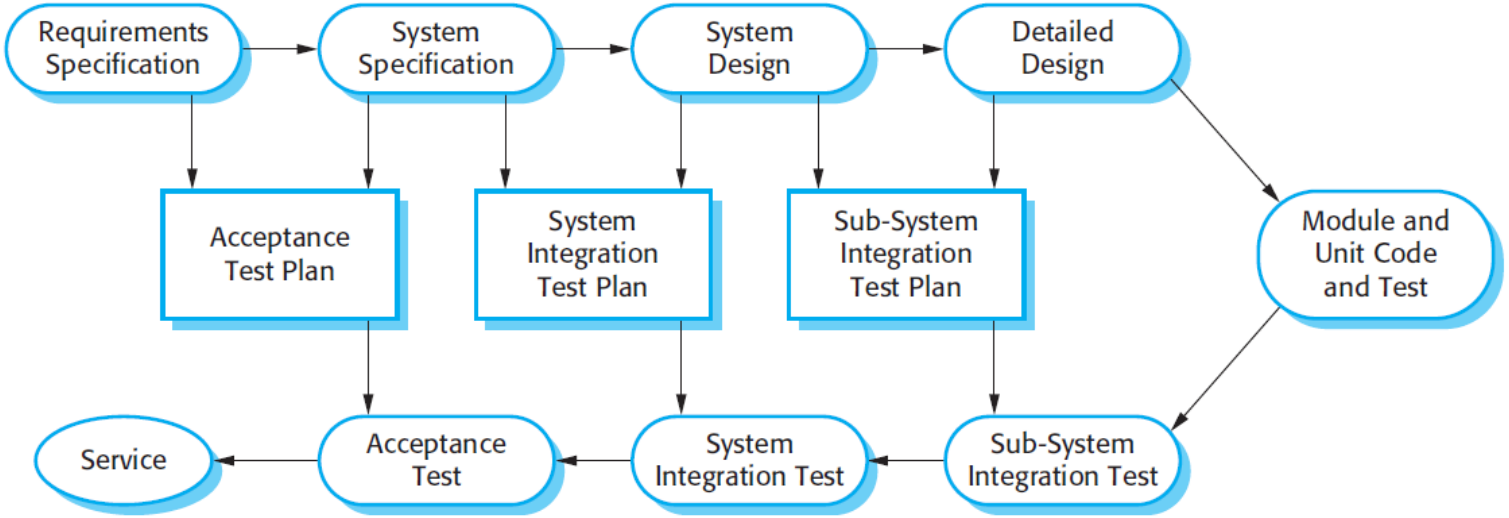


กิจกรรมในกระบวนการ (Cont.)

การตรวจสอบซอฟต์แวร์ (Software Validation)

- การตรวจสอบซอฟต์แวร์ เป็นส่วนหนึ่งของเทคนิคการทวนสอบและตรวจสอบ (verification and validation: V&V) มีวัตถุประสงค์เพื่อแสดงให้เห็นว่าระบบมีความสอดคล้องตามข้อกำหนดและตรงกับความคาดหวังหรือความต้องการของระบบที่ลูกค้าต้องการ
  - การทดสอบโปรแกรม (Program testing)
  - การตรวจสอบความถูกต้อง (Validation)

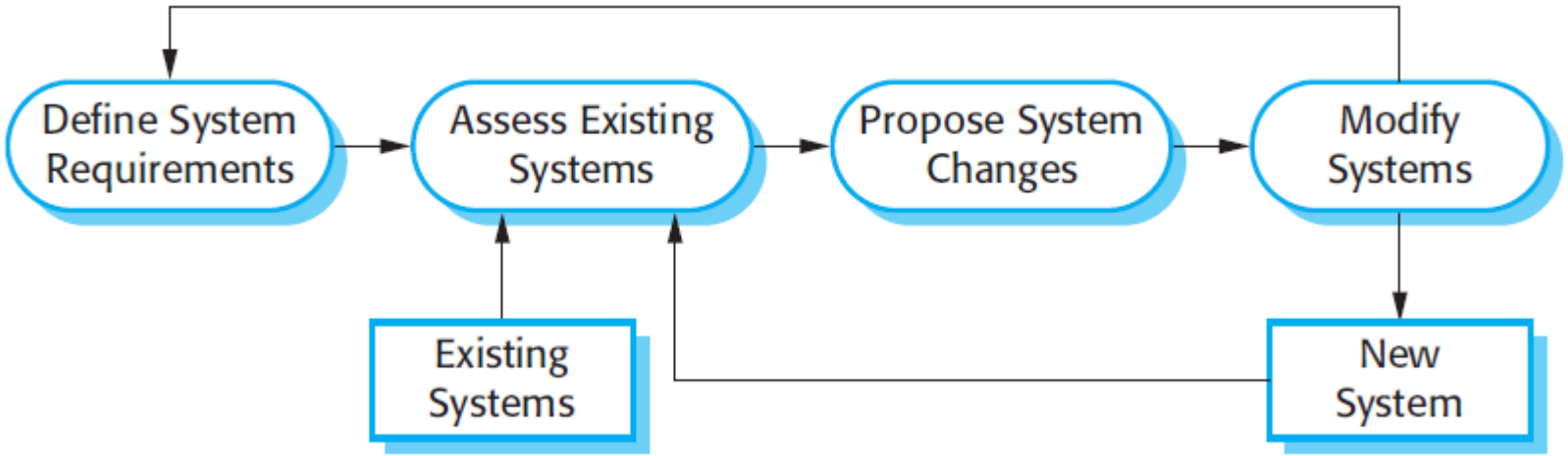




กิจกรรมในกระบวนการ (Cont.)

การปรับปรุงและบำรุงรักษาซอฟต์แวร์ (Software Evolution)

- เป็นหนึ่งในกิจกรรมที่ยืดหยุ่นของระบบซอฟต์แวร์ ซึ่งเป็นสาเหตุหลักที่ทำให้ซอฟต์แวร์ถูกรวมเป็นระบบขนาดใหญ่และมีความซับซ้อน
- วิศวกรรมซอฟต์แวร์มีความเกี่ยวข้องกับกระบวนการปรับปรุงซอฟต์แวร์ที่มีการเปลี่ยนแปลงเกิดขึ้นในตลอดช่วงอายุการใช้งาน เพื่อตอบสนองความต้องการที่เปลี่ยนแปลงและความจำเป็นของลูกค้า



กระบวนการพัฒนาซอฟต์แวร์ (ต่อ)

ผลิตภัณฑ์ (Products)

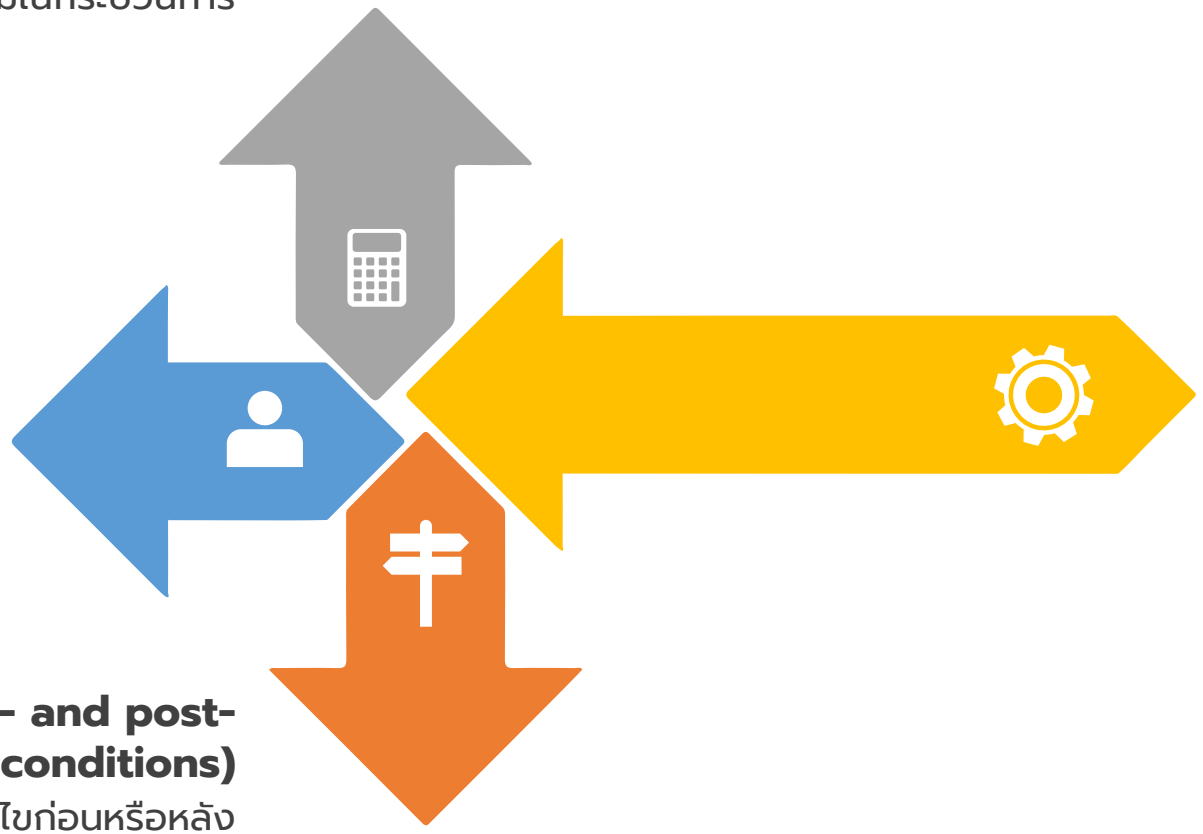
ผลลัพธ์ของกิจกรรมในกระบวนการ

บทบาท (Roles)

บทบาทความรับผิดชอบของผู้ที่มีส่วนเกี่ยวข้องในแต่ละกระบวนการ

เงื่อนไขก่อนหรือหลัง (Pre- and post-conditions)

คำสั่งที่เป็นข้อเท็จจริงซึ่งเป็นเงื่อนไขก่อนหรือหลังกิจกรรมในกระบวนการต่าง ๆ ที่จะถูกนำไปใช้งานหรือผลิตเป็นผลิตภัณฑ์



เมื่อมีการอธิบายหรืออภิปรายถึงกระบวนการต่าง ๆ ตลอดจนลำดับของกิจกรรมที่เกิดขึ้น

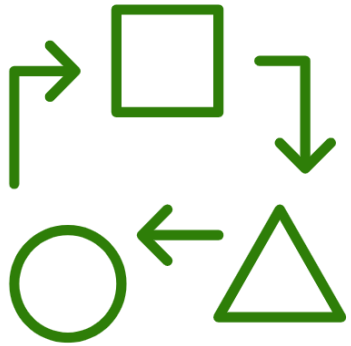
อย่างไรก็ตาม กิจกรรมและรายละเอียดของกระบวนการต่าง ๆ อาจรวมถึงองค์ประกอบ ดังนี้

กระบวนการพัฒนาซอฟต์แวร์ (ต่อ)

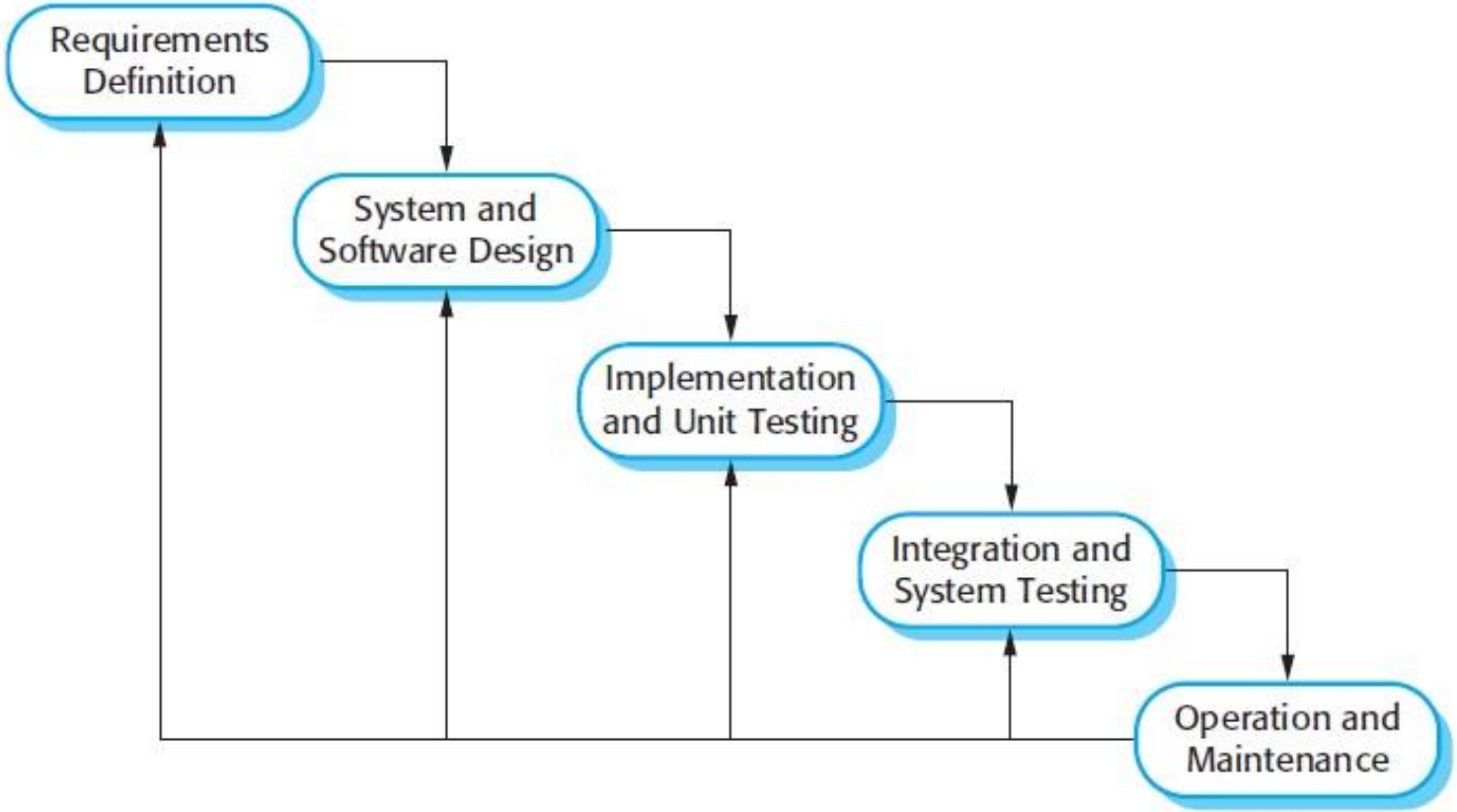
- กิจกรรมที่เกิดขึ้นในกระบวนการพัฒนาซอฟต์แวร์ สำหรับแต่ละกิจกรรมมีความแตกต่างกันอย่างไร?

แบบจำลองกระบวนการพัฒนาซอฟต์แวร์

- แบบจำลองกระบวนการพัฒนาซอฟต์แวร์ (Software process model)
  - การจำลองภาพของกระบวนการพัฒนาซอฟต์แวร์ เพื่อให้เห็นถึงการจัดโครงสร้างลำดับขั้นตอนของกระบวนการที่แตกต่างกัน
  - พื้นฐานแบบจำลองกระบวนการผลิตซอฟต์แวร์มี 3 รูปแบบ ดังนี้
    - แบบจำลองน้ำตก (Waterfall Model)
    - การพัฒนาแบบเพิ่มเติม (Incremental Development)
    - วิศวกรรมซอฟต์แวร์เชิงการนำกลับมาใช้ใหม่ (Reuse-Oriented Software Engineering)



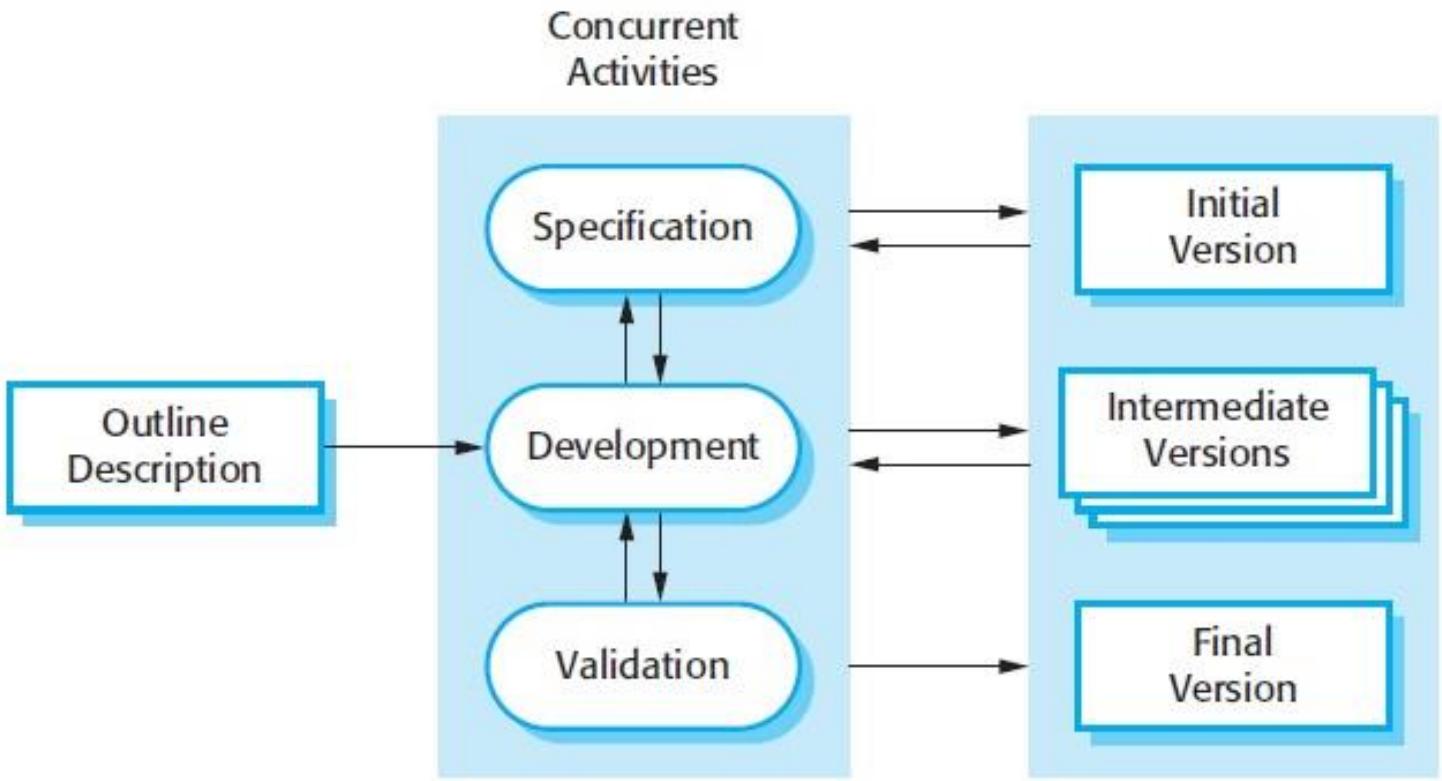
แบบจำลองน้ำตก (Waterfall model)



แบบจำลองน้ำตกนำมาใช้เป็นกระบวนการพื้นฐาน ซึ่งมีกิจกรรม ดังนี้ การกำหนดคุณสมบัติ (specification), การพัฒนา (development), การตรวจสอบ (validation), และ การปรับปรุง (evolution) และแทนด้วยขั้นตอนต่าง ๆ ของกระบวนการที่แยกจากกัน ได้แก่ การกำหนดความต้องการ (requirements specification), การออกแบบซอฟต์แวร์ (software design), การนำเสนอ (implementation), การทดสอบ (testing) และอื่น ๆ

การพัฒนาแบบเพิ่มเติม (Incremental Development)

แนวทางนี้เป็นแนวทางที่สอดแทรกในทุก ๆ กิจกรรมของการกำหนด (specification), การพัฒนา (development) และ การตรวจสอบ (validation) โดยระบบที่ถูกพัฒนาขึ้นมาเป็นชุดของเวอร์ชัน (increments) ที่ซึ่งในแต่ละเวอร์ชันจะเป็นการเพิ่มเติมฟังก์ชันการทำงานมาจากเวอร์ชันก่อนหน้า

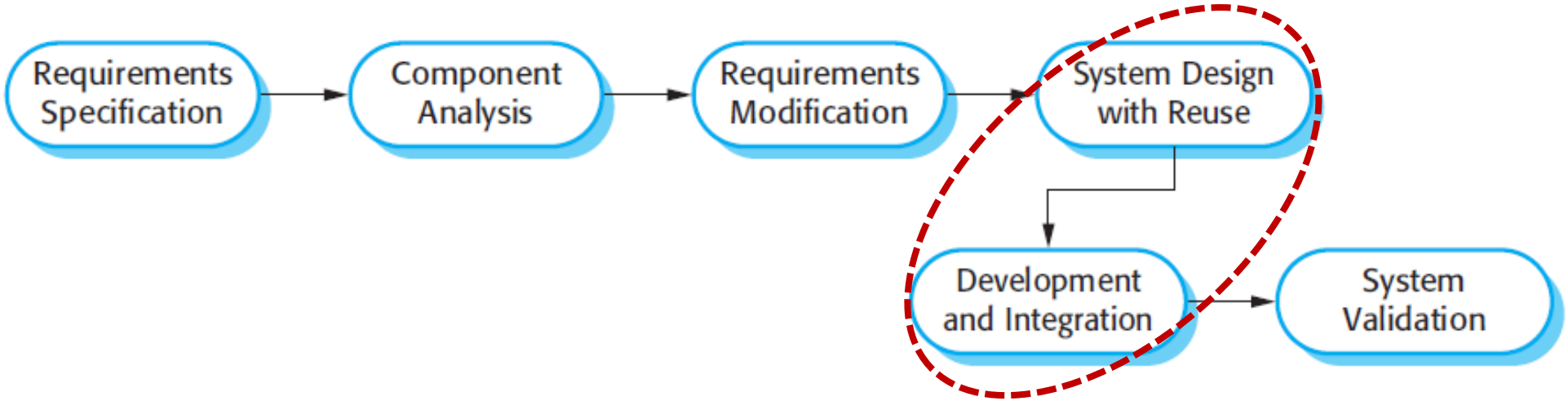


การพัฒนาแบบเพิ่มเติมนั้น มีประโยชน์ที่สำคัญ ๆ 3 ประการ เมื่อเปรียบเทียบกับแบบจำลองน้ำตก:

1. ค่าใช้จ่ายที่เกิดขึ้นจากการเปลี่ยนแปลงความต้องการของลูกค้าลดลง
2. ง่ายต่อการรับร้องเรียนหรือติชมจากลูกค้า ส่งผลให้การพัฒนางานแล้วเสร็จทันเวลาที่กำหนด
3. สามารถส่งมอบซอฟต์แวร์ได้เร็วและซอฟต์แวร์ที่พัฒนาเอื้อประโยชน์ต่อลูกค้าแม้ว่าจะได้รวมฟังก์ชันการทำงานของซอฟต์แวร์ไว้ทั้งหมด

วิศวกรรมซอฟต์แวร์เชิงการนำกลับมาใช้ใหม่ (Reuse-Oriented Software Engineering)

แนทางนี้อยู่บนพื้นฐานของการมีอยู่ของจำนวนองค์ประกอบที่สามารถนำมาใช้ซ้ำได้ โดยกะบวนการพัฒนาระบบมุ่งเน้นที่การผสานการทำงานระหว่างองค์ประกอบเหล่านั้นกับระบบที่มีอยู่มากกว่าการพัฒนาระบบเริ่มต้นใหม่

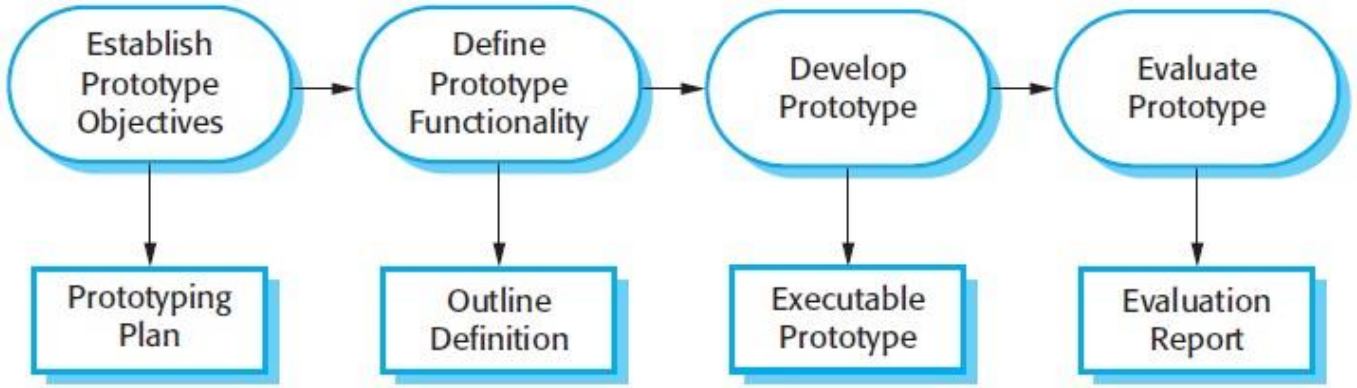


องค์ประกอบของซอฟต์แวร์ 3 ประเภท ที่ซึ่งสามารถนำมาใช้ในกระบวนการเชิงนำกลับมาใช้ใหม่ ประกอบด้วย ดังนี้
1. เว็บเซอร์วิส (Web services)
2. แหล่งรวมวัตถุ (Collections of objects), เฟรมเวิร์คคอมโพเนนต์ (component framework) เช่น .NET หรือ J2EE.
3. ซอฟต์แวร์ระบบแบบเดี่ยว (Stand-alone software systems)

แบบจำลองกระบวนการพัฒนาซอฟต์แวร์ (ต่อ)

- ปัจจุบันมีการปรับกระบวนการผลิตซอฟต์แวร์ให้มีประสิทธิภาพมากขึ้น จึงทำให้แบบจำลองมีรูปแบบต่างกัน ดังนี้
  - แบบจำลองโปรโตไทป์ (Prototyping)
  - แบบจำลองการส่งมอบเพิ่มเติม (Incremental delivery)
  - แบบจำลองโบเอ็มสไปรัล (Boehm Spiral Model: BSM)
  - แบบจำลองกระบวนการ Rational Unified (The Rational Unified Process: RUP)

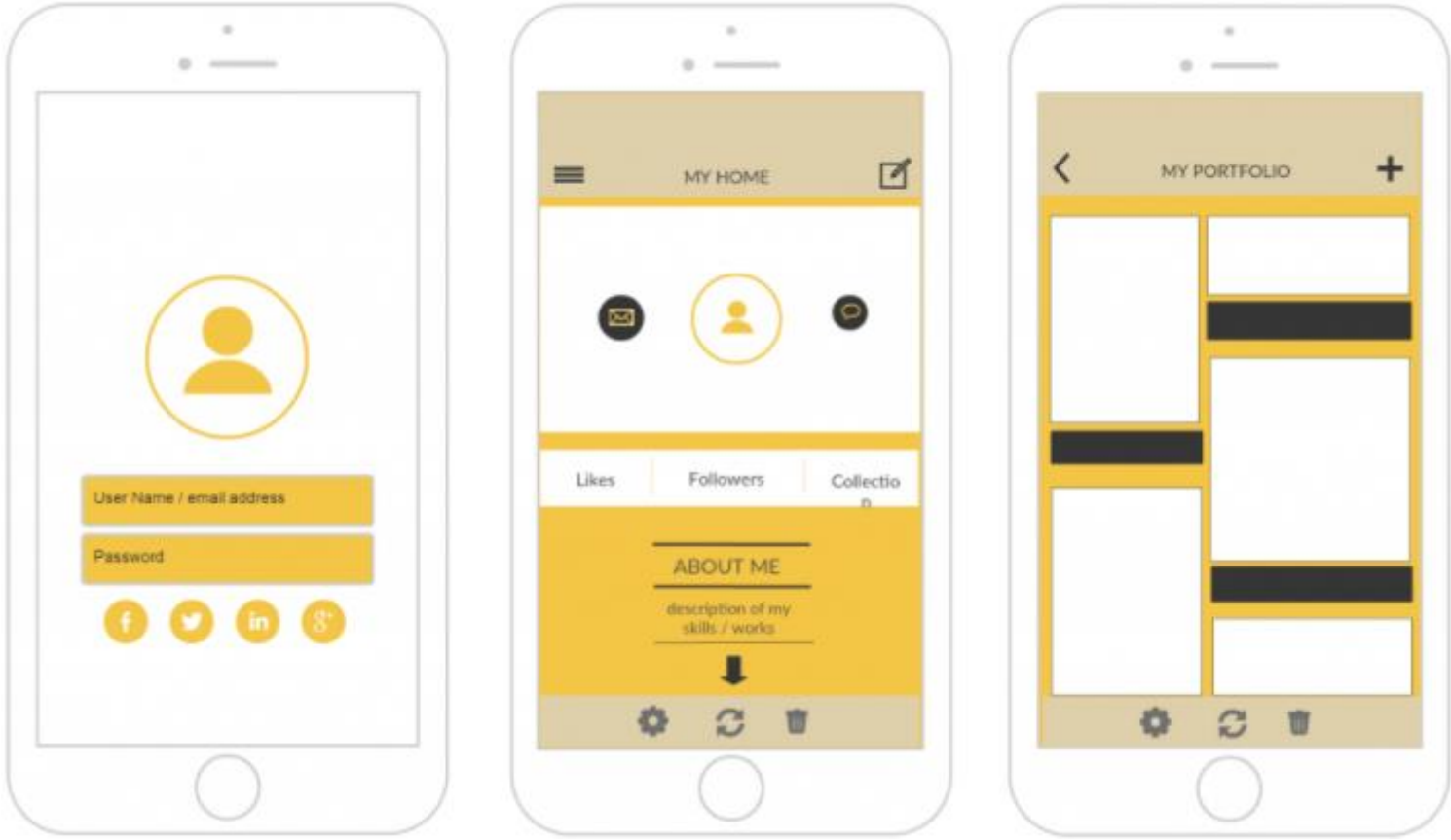
แบบจำลองโปรโตไทป์ (Prototyping)



- โปรโตไทป์ซอฟต์แวร์ สามารถนำไปใช้ในกระบวนการพัฒนาซอฟต์แวร์เพื่อช่วยคาดการณ์การเปลี่ยนแปลงที่จำเป็นซึ่งอาจจะเกิดขึ้น ดังนี้
  - ในกระบวนการวิศวกรรมความต้องการ โปรโตไทป์สามารถช่วยสกัดและตรวจสอบความต้องการของระบบ
  - ในกระบวนการออกแบบระบบ โปรโตไทป์สามารถช่วยในการสำรวจวิธีการสำหรับการช่วยแก้ปัญหาของซอฟต์แวร์โดยเฉพาะ และยังช่วยสนับสนุนการออกแบบส่วนต่อประสานผู้ใช้

- ข้อจำกัดของแบบจำลอง ดังนี้
  - อาจเป็นไปไม่ได้ที่จะปรับแต่งต้นแบบให้ตรงตามความต้องการไม่ใช่เชิงหน้าที่ (Non-Functional Requirements)
  - การเปลี่ยนแปลงที่เกิดขึ้นอย่างรวดเร็วในระหว่างการพัฒนาด้วยโปรโตไทป์จะไม่มีเอกสาร คำสั่งโปรโตไทป์ที่พัฒนาขึ้นมาอาจจะไม่ดีพอสำหรับการบำรุงรักษาในระยะยาว
  - การเปลี่ยนแปลงที่เกิดขึ้นในระหว่างการพัฒนาโปรโตไทป์นั้น อาจส่งผลต่อโครงสร้างของระบบที่ไม่ดี ซึ่งอาจจะยากต่อการบำรุงรักษาและมีค่าใช้จ่ายที่ค่อนข้างสูง
  - มาตรฐานด้านคุณภาพขององค์กรอาจจะไม่เข้มงวดนัก เมื่อพัฒนาระบบด้วยแบบจำลองโปรโตไทป์

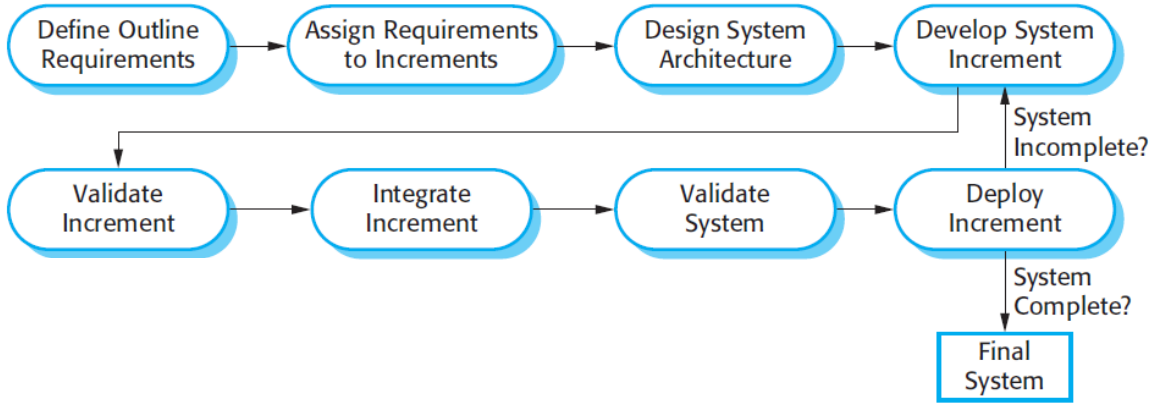
แบบจำลองโปรโตไทป์ (ต่อ)



Source: https://creately.com/blog/diagrams/iphone-mockup-templates/

แบบจำลองการส่งมอบเพิ่มเติม (Incremental delivery)

แนวทางในการพัฒนาซอฟต์แวร์ด้วยแบบจำลองนี้ โดยบางฟังก์ชันที่พัฒนาเพิ่มเติมนั้นจะถูกส่งมอบให้กับลูกค้าเพื่อนำไปติดตั้งและใช้งานภายใต้สภาพแวดล้อมสำหรับการปฏิบัติงานจริง



แบบจำลองการส่งมอบเพิ่มเติมมีข้อดี ดังนี้

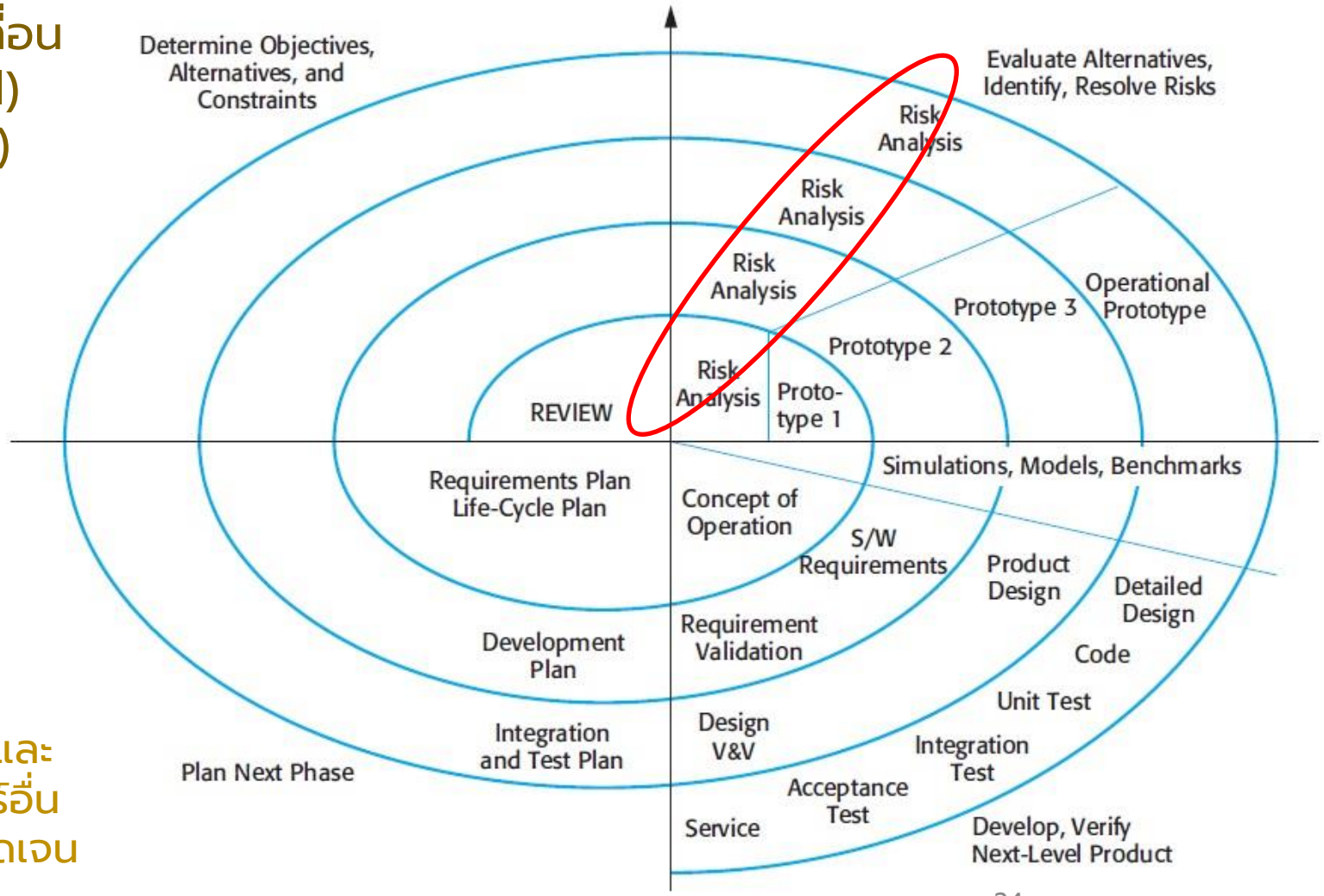
- ลูกค้าสามารถนำส่วนที่เพิ่มเติมขึ้นมาในระยะเริ่มต้นไปใช้เป็นต้นแบบและเป็นประสบการณ์ในการบอกหรือแนะนำความต้องการสำหรับการเพิ่มในระบบภายหลัง ซึ่งต่างจากโปรโตไทป์ เนื่องจากองค์ประกอบเหล่านั้นถูกรวมไว้ในระบบจริงแล้วเมื่อระบบทั้งหมดพร้อมใช้งานแล้วจึงอาจจะไม่ได้เรียนรู้จากการทำซ้ำ
- ลูกค้าไม่ต้องรอจนกว่าระบบทั้งหมดจะพัฒนาเสร็จจึงจะส่งมอบ แต่สามารถได้รับประโยชน์จากระบบในระยะแรก ๆ เนื่องจากการเพิ่มเติมดังกล่าวเป็นไปตามข้อกำหนดที่สำคัญ ๆ เพื่อให้ซอฟต์แวร์สามารถใช้งานได้ทันที
- การพัฒนาแบบเพิ่มเติมเอื้อประโยชน์ต่อกระบวนการบำรุงรักษา ซึ่งจะง่ายต่อการรวมการเปลี่ยนแปลงส่วนต่าง ๆ เข้ากับระบบ
- เซอร์วิสที่มีความสำคัญลำดับต้น ๆ จะถูกนำเสนอและส่งมอบก่อนเป็นอันดับแรกและถูกทดสอบมากที่สุด จากนั้นค่อยรวมระบบ

แบบจำลองการส่งมอบเพิ่มเติม (ต่อ)

- **ปัญหาจากการพัฒนาด้วยแบบจำลองการส่งมอบเพิ่มเติม**
  - โดยส่วนใหญ่ระบบมีความจำเป็นในการใช้ชุดสิ่งอำนวยความสะดวกพื้นฐาน ที่นำไปใช้ในส่วนต่าง ๆ ของระบบที่แตกต่างกัน
  - การพัฒนาแบบทำซ้ำ อาจจะเป็นเรื่องที่ยากเมื่อมีการพัฒนาระบบใหม่มาทดแทน เนื่องจากข้อกำหนดต่าง ๆ ที่พัฒนาขึ้นมานั้นจะรวมเข้ากับซอฟต์แวร์

แบบจำลองบเอ็มสไปรัล (Boehm Spiral Model: BSM)

กรอบกระบวนการซอฟต์แวร์ที่ขับเคลื่อนด้วยความเสี่ยง (the spiral model) เสนอโดย Boehm (ใน ปี ค.ศ. 1988)



ความแตกต่างระหว่างแบบจำลองสไปรัลและแบบจำลองกระบวนการพัฒนาซอฟต์แวร์อื่น คือ มีการรับรู้เกียวกับความเสี่ยงอย่างชัดเจน

แบบจำลองบเอ็มสไปรัล (ต่อ)

โดยสไปรัลจำแนกเป็น 4 ส่วน ดังนี้

- ***การกำหนดวัตถุประสงค์ (Objective setting)***
  - วัตถุประสงค์สำหรับการดำเนินการในแต่ละขั้นตอนในโครงการต้องกำหนดหรือนิยาม
- ***การประเมินและลดความเสี่ยง (Risk assessment and reduction)***
  - สำหรับการระบุความเสี่ยงในแต่ละโครงการ มีการวิเคราะห์โดยละเอียดและดำเนินการตามขั้นตอนเพื่อลดความเสี่ยง
- ***การพัฒนาและการตรวจสอบ (Development and validation)***
  - หลังจากประเมินความเสี่ยงแล้วแบบจำลองการพัฒนาจะถูกเลือกเพื่อใช้ในการพัฒนาระบบ
- ***การวางแผน (Planning)***
  - โครงการต้องได้รับการทวนสอบและตัดสินใจว่าจะดำเนินการต่อเนื่องในวงรอบของสไปรัลต่อหรือไม่
  - หากตัดสินใจที่จะดำเนินการต่อ ต้องมีการร่างแผนการดำเนินการในขั้นตอนถัดไปของโครงการ

แบบจำลองกระบวนการ Rational Unified (RUP)

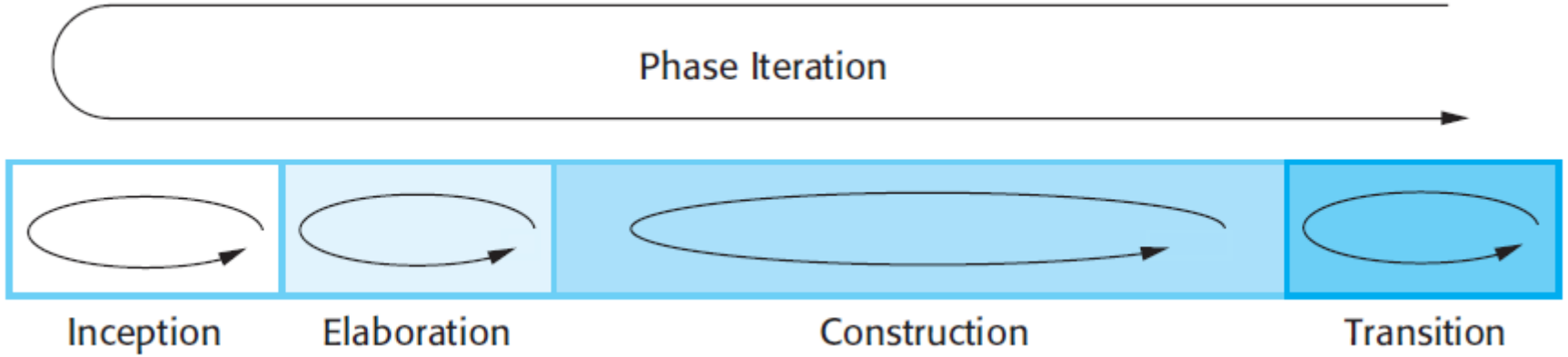
- แบบจำลองกระบวนการ Rational Unified (RUP) (Krutchen, 2003)
  - เป็นหนึ่งในตัวอย่างของแบบจำลองกระบวนการรูปแบบใหม่ ที่ได้จากการทำงานด้วย UML และการทำงานแบบรวมศูนย์ที่เชื่อมโยงกันของกระบวนการพัฒนาซอฟต์แวร์
  - **แบบจำลองกระบวนแบบไฮบริด (Hybrid process model)**
    - แบบจำลองกระบวนการโดยทั่วไป ประกอบด้วยการกำหนดคุณสมบัติและการออกแบบสามารถสนับสนุนการพัฒนาโปรโตไทป์และการส่งมอบเพิ่มเติม
- โดย RUP สามารถอธิบายด้วย 3 มุมมอง ดังนี้

มุมมองไดนามิก (A dynamic perspective) แสดงถึงขั้นตอนในแบบจำลองที่เกิดขึ้นในแต่ละช่วงเวลา

มุมมองคงที่ (A static perspective) แสดงถึงกิจกรรมในกระบวนการที่กำหนดขึ้น

มุมมองเชิงฏิบัติ (A practice perspective) ที่ซึ่งแนะนำแนวปฏิบัติที่ดีที่ใช้ในกระบวนการ

แบบจำลองกระบวนการ Rational Unified (ต่อ)



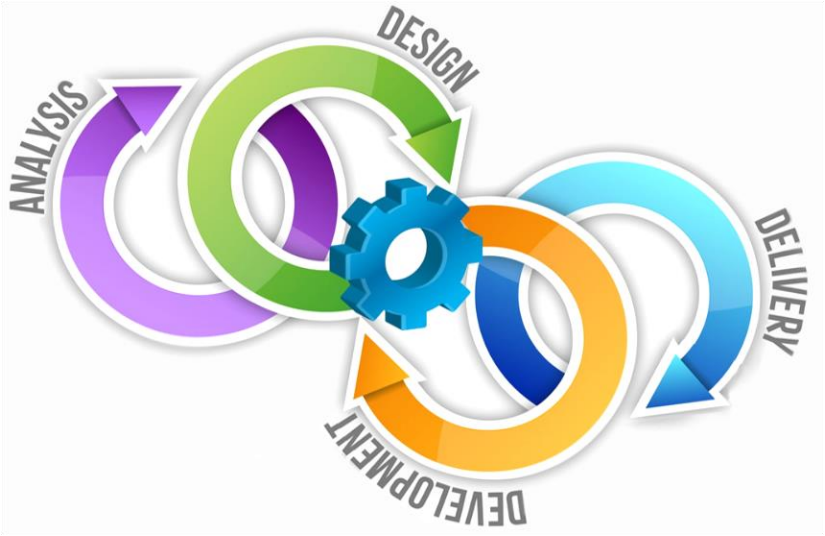
- โดยแนวปฏิบัติเบื้องต้นที่แนะนำ ประกอบด้วย 6 แนวปฏิบัติ ดังนี้
  - การพัฒนาซอฟต์แวร์แบบวนซ้ำ (Develop software iteratively)
  - การจัดการความต้องการ (Manage requirements)
  - การใช้สถาปัตยกรรมตามคอมโพเนนต์พื้นฐาน (Use component-based architectures)
  - ซอฟต์แวร์แบบจำลองภาพ (Visually model software)
  - คุณภาพการทวนสอบซอฟต์แวร์ (Verify software quality)
  - การควบคุมการเปลี่ยนแปลงซอฟต์แวร์ (Control changes to software)

เวิร์คโฟลว์แบบคงที่ใน RUP (Static Workflows in RUP)

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models, and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users, and installed in their workplace.
Configuration and change management	This supporting workflow manages changes to the system (see Chapter 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

แนวทางการประยุกต์ใช้กระบวนการพัฒนาซอฟต์แวร์

- กระบวนการพัฒนาซอฟต์แวร์มีขั้นตอนที่สำคัญ
  - .....
- ก่อนเริ่มโครงการ วิศวกรซอฟต์แวร์ควรพิจารณาเลือกแบบจำลองที่เหมาะสม
- โดยต้องพิจารณาปัจจัยต่าง ๆ ดังนี้
  - ความต้องการ ความเสี่ยง เทคนิคการใช้งาน ขอบเขตองค์ความรู้ และจุดมุ่งเน้น



สรุป (Summary)

- กระบวนการพัฒนาซอฟต์แวร์
  - กิจกรรมในกระบวนการ (Process Activities)
- แบบจำลองกระบวนการพัฒนาซอฟต์แวร์
- แนวทางการประยุกต์ใช้กระบวนการพัฒนาซอฟต์แวร์

กิจกรรมท้ายบท

- กำหนดให้บอกเหตุผลบนพื้นฐานของประเภทของระบบที่กำลังพัฒนา โดยแนะนำรูปแบบของกระบวนการพัฒนาซอฟต์แวร์ทั่วไป ที่ซึ่งเหมาะสมที่สุดเพื่อใช้เป็นพื้นฐานในการจัดการพัฒนาระบบ ดังนี้
  - ระบบควบคุมการเบรกโดยป้องกันล้อล็อก (Anti-lock Braking) ในรถยนต์
  - ระบบเสมือนจริง (Virtual reality system) เพื่อสนับสนุนการบำรุงรักษาซอฟต์แวร์
  - ระบบบัญชีของมหาวิทยาลัย (University accounting system) ที่ซึ่งจะนำมาแทนระบบเดิม
  - ระบบวางแผนการเดินทางแบบโต้ตอบ (Interactive travel planning system) ที่ช่วยให้ผู้ใช้งานวางแผนการเดินทางโดยกระทบต่อการจัดการสิ่งแวดล้อมหรือปัจจัยที่เกี่ยวข้องต่าง ๆ น้อยที่สุด

เอกสารอ้างอิง

- กิตติ ภักดีวัฒนะกุล, วิศวกรรมซอฟต์แวร์ (Software Engineering), กรุงเทพฯ: เคทีพี คอมพ์ แอนด์ คอนซัลท์, 2552
- น้ำฝน อัศวเมฆิน, หลักการพื้นฐานของวิศวกรรมซอฟต์แวร์ (Fundamentals of Software Engineering), กรุงเทพฯ: ซีเอ็ดยูเคชั่น, 2560.
- Lan Sommerville, Software Engineering Ninth Edition, Pearson Education, Inc., publishing as Addison-Wesley, 2011.
- Roger S. Pressman and Bruce R. Maxim, Software Engineering: A Practitioner's Approach, Eighth Edition, McGraw-Hill Education, 2015.