บทที่ 2

กระบวนการซอฟต์แวร์

(Software Process)

สอนโดย **อ.พนารัตน์ ศรีเชษฐา**

สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ

คณะวิทยาศาสตร์ มหาวิทยาลัยราชภัฏอุดรธานี

วัตถุประสงค์การเรียนรู้

- เพื่อแนะนำแนวคิดของกระบวนการซอฟต์แวร์และแบบจำลองกระบวนการซอฟต์แวร์
- เพื่ออธิบายกิจกรรมต่างๆ ที่เกี่ยวข้องกับวิศวกรรมความต้องการทางซอฟต์แวร์, การพัฒนาซอฟต์แวร์, การทดสอบและการปรับปรุงให้ดีขึ้น
- เพื่ออธิบายแบบจำลองกระบวนการซอฟต์แวร์
- เพื่ออธิบายแบบจำลองของ Rational Unified Process

หัวข้อการเรียนรู้

- กระบวนการซอฟต์แวร์ (Software process)
- แบบจำลองกระบวนการซอฟต์แวร์ (Software process models)
- การวนรอบกระบวนการ (Process iteration)
- กระบวนการรวมเป็นหนึ่งเดียวแบบมีเหตุผล (Rational Unified Process)
- เครื่องมือและระเบียบวิธีการทางวิศวกรรมซอฟต์แวร์ (CASE Tools and Methodology in Software Engineering)

♦ กระบวนการซอฟต์แวร์ (Software process)

- **กระบวนการซอฟต์แวร์** หมายถึง ชุดของกิจกรรมที่จำเป็นสำหรับการพัฒนาซอฟต์แวร์
- **กิจกรรมทั่วไปของกระบวนการซอฟต์แวร์**
  - การกำหนดคุณลักษณะของซอฟต์แวร์ (Software Specification)
  - การออกแบบและสร้างซอฟต์แวร์ (Software Design and Implementation)
  - การตรวจสอบซอฟต์แวร์ (Software Validation)
  - การวิวัฒนาการซอฟต์แวร์ (Software Evolution)
- ใน waterfall model กิจกรรมเหล่านี้ทำตามลำดับก่อนหลัง
- ใน evolutionary process กิจกรรมเหล่านี้เกี่ยวพันกันอยู่

กระบวนการซอฟต์แวร์ (Software process)

1. **การกำหนดคุณลักษณะซอฟต์แวร์ (Software Specification)**

   : กิจกรรมกำหนดหน้าที่ต่างๆ ที่ต้องมีในซอฟต์แวร์ และระบุข้อจำกัด กฎระเบียบต่างๆ ที่เกี่ยวข้องกับกระบวนการพัฒนาซอฟต์แวร์

2. **การออกแบบและสร้างซอฟต์แวร์ (Software Design and Implementation)**

   : กิจกรรมการออกแบบและสร้างซอฟต์แวร์ ให้ตรงกับข้อกำหนด (specification) ของซอฟต์แวร์

3. **การตรวจสอบซอฟต์แวร์ (Software Validation)**

   : กิจกรรมการออกแบบตรวจสอบความถูกต้องของซอฟต์แวร์ เพื่อให้เกิดความมั่นใจว่า ซอฟต์แวร์ที่ผลิตขึ้นได้ตรงกับความต้องการของลูกค้า

4. **การวิวัฒนาการซอฟต์แวร์ (Software Evolution)**

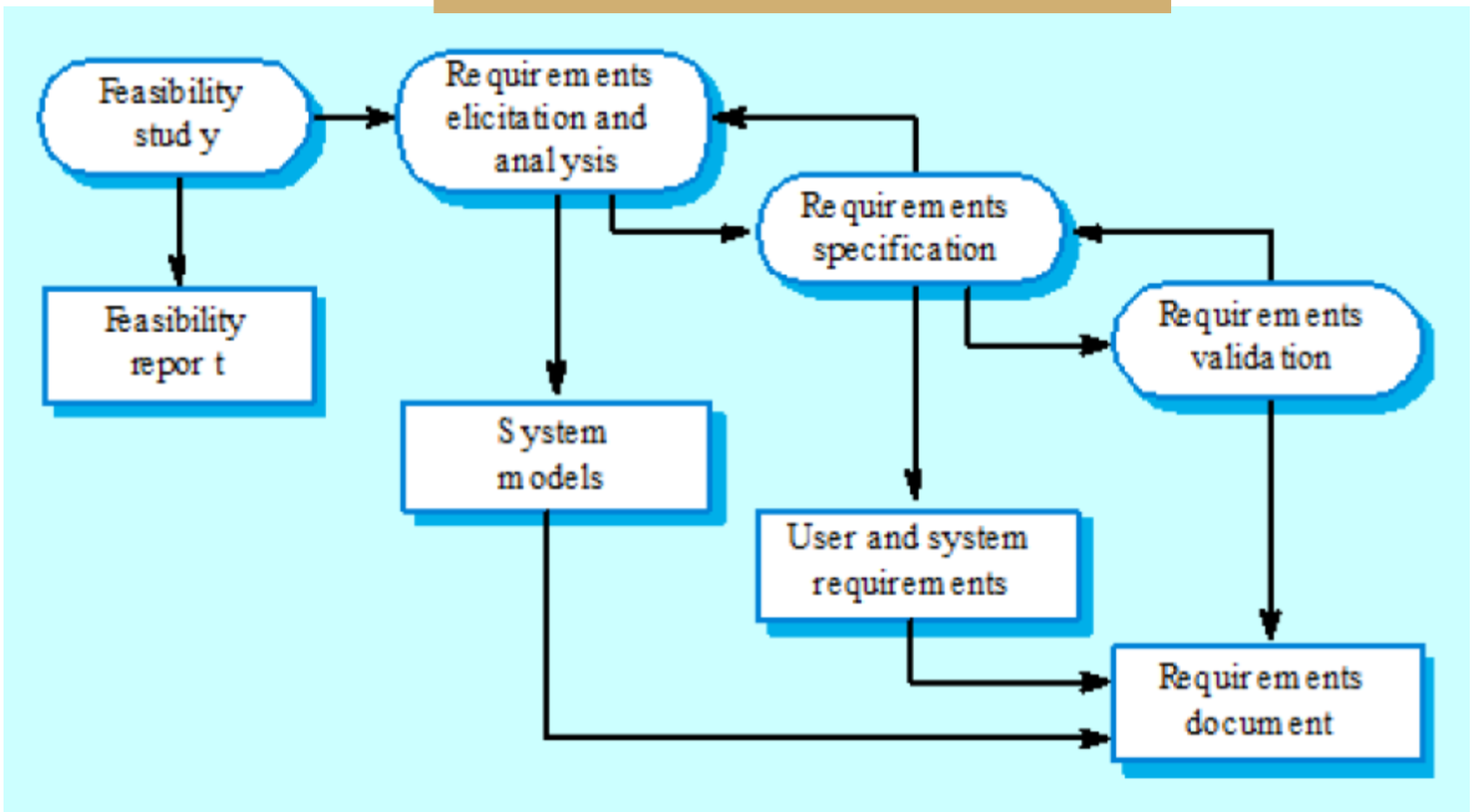
   : กิจกรรมการเตรียมการบางอย่าง เพื่อจัดการกับเหตุการณ์ที่คาดหมายว่าจะเกิดขึ้นในอนาคต หลังจากลูกค้าใช้งานซอฟต์แวร์ที่ผลิตไประยะหนึ่งแล้ว เช่น การเปลี่ยนแปลงความต้องการ หรือการมีความต้องการเพิ่มเติม

1. การกำหนดคุณลักษณะซอฟต์แวร์ (software specification)

- เป็นกิจกรรมที่กำหนดความต้องการที่ต้องมีให้กับระบบ และระบุข้อจำกัด กฎระเบียบต่างๆ ที่เกี่ยวข้องกับกระบวนการพัฒนาซอฟต์แวร์
- บางครั้งเรียกกิจกรรมนี้ว่า วิศวกรรมความต้องการ (Requirements Engineering)
- **กระบวนการวิศวกรรมความต้องการ (Requirements engineering process)** ประกอบด้วย
  - ศึกษาความเป็นไปได้
  - ค้นหาและวิเคราะห์ความต้องการ
  - กำหนดรายละเอียดให้กับความต้องการ
  - ยืนยันความต้องการ
- ผลลัพธ์ของกิจกรรมนี้คือ เอกสารความต้องการ (Requirements Documents)

1. การกำหนดคุณลักษณะซอฟต์แวร์

The requirements engineering process

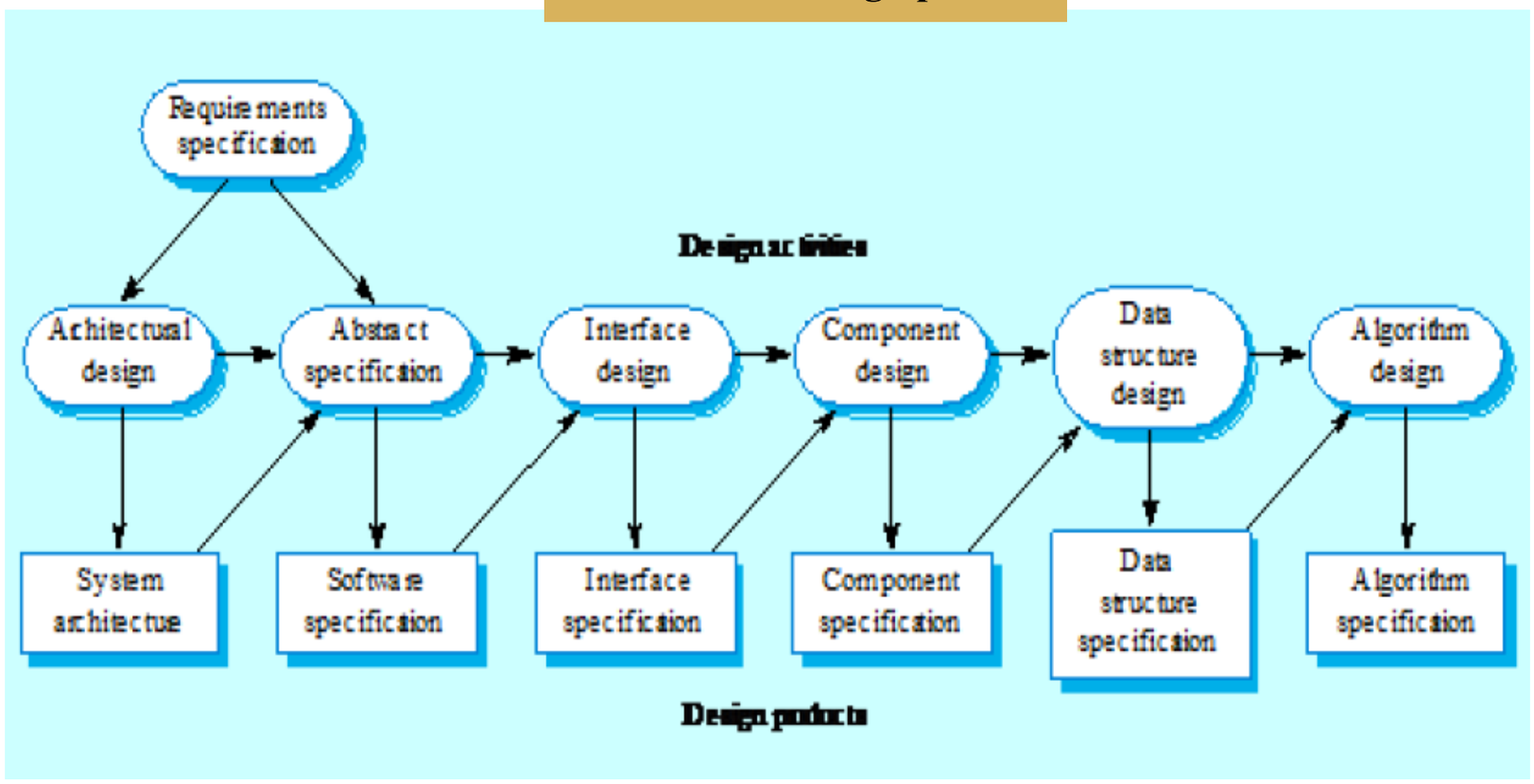


2. การออกแบบและสร้างซอฟต์แวร์ software design and implementation

- **Software Design** เป็นกิจกรรมการออกแบบซอฟต์แวร์ ให้สัมพันธ์กับข้อกำหนด ของซอฟต์แวร์ และสร้างซอฟต์แวร์ให้ทำงานได้ตามสิ่งที่ได้ออกแบบเอาไว้
- **Software Implementation** เป็นกิจกรรมการเปลี่ยนข้อกำหนดของระบบ ให้เป็นระบบที่สามารถทำงานได้ (Executable system)
- **กิจกรรมของการออกแบบซอฟต์แวร์** ประกอบด้วย
  - การออกแบบสถาปัตยกรรม (Architectural design)
  - การระบุข้อกำหนดนามธรรม (Abstract specification)
  - การออกแบบส่วนติดต่อ (Interface design)
  - การออกแบบส่วนประกอบ (Component design)
  - การออกแบบโครงสร้างข้อมูล (Data structure design)
  - การออกแบบอัลกอริทึ่ม (Algorithm design)

2. การออกแบบและสร้างซอฟต์แวร์

The software design process

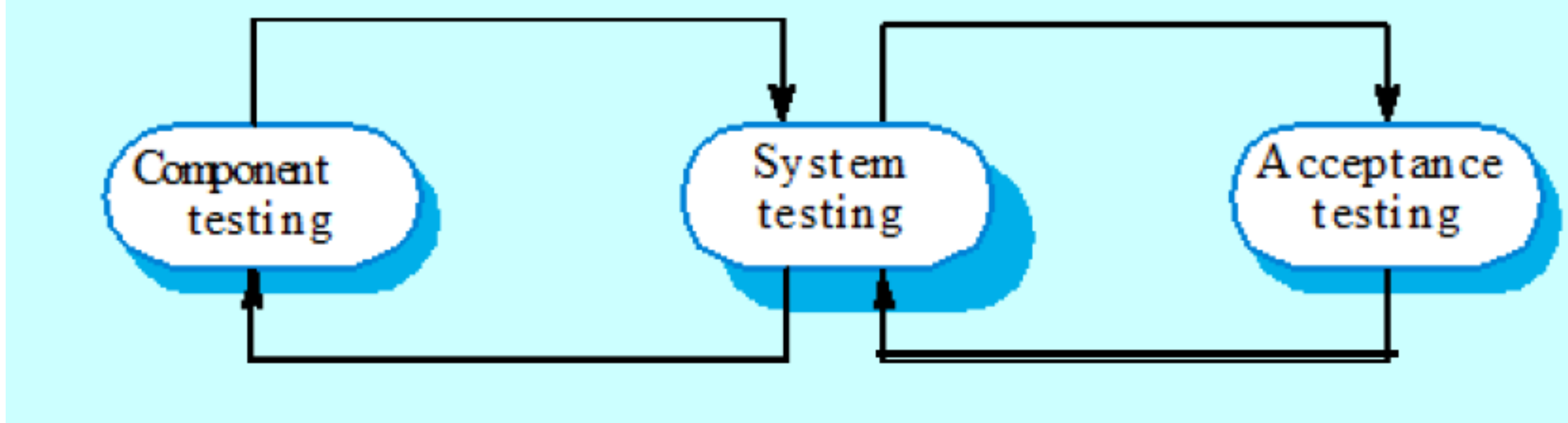


3. การตรวจสอบซอฟต์แวร์ software validation

- รูปแบบทั่วไปของการตรวจสอบซอฟต์แวร์ คือ **Validation and Verification (V & V)**
- **V & V** คือ การตรวจเช็คและกระบวนการทบทวนและทดสอบระบบ มีไว้เพื่อแสดงให้เห็นว่าระบบได้ทำขึ้นมาตามที่กำหนดและได้ตามความต้องการของลูกค้า

3. การตรวจสอบซอฟต์แวร์

The Testing Process



การทดสอบคอมโพเนนท์ เป็นการทดสอบระดับหน่วยหรือคอมโพเนนท์แบบอิสระ ว่าทำงานได้ถูกต้องหรือไม่

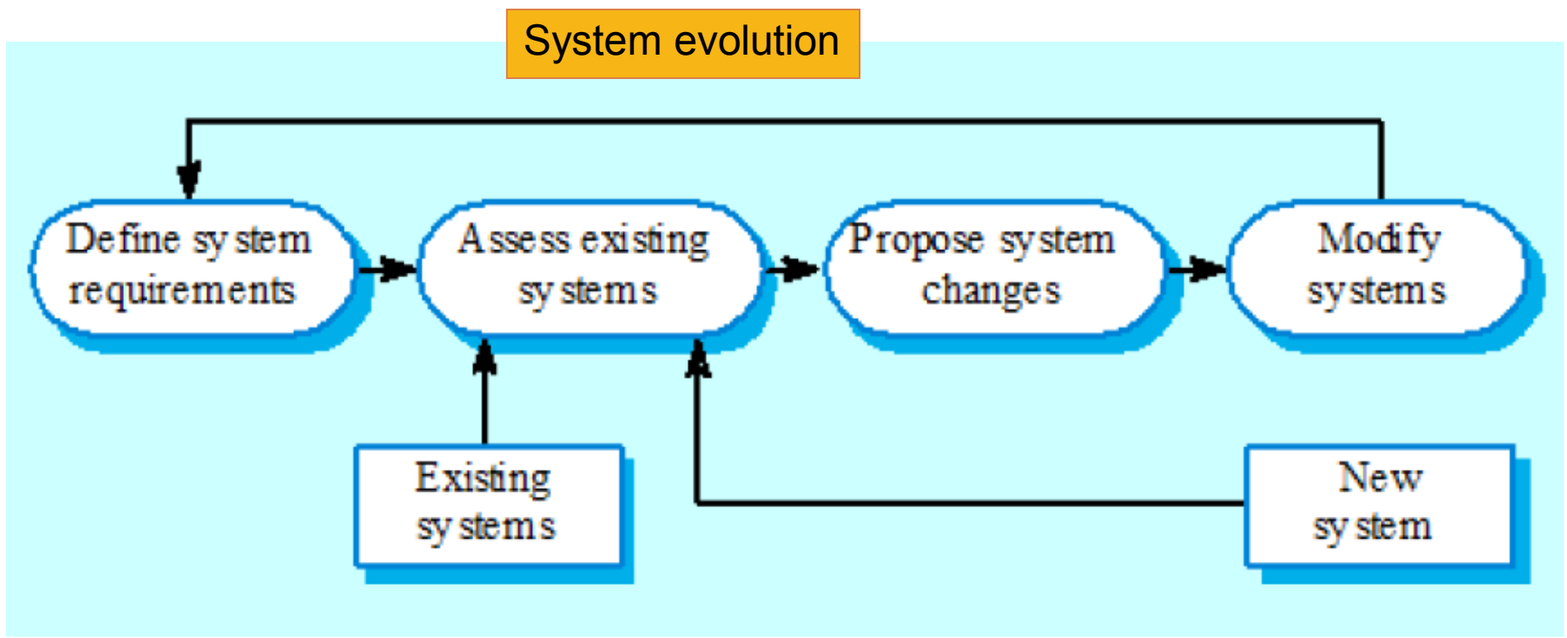
การทดสอบระบบ ซึ่งระบบเกิดจากการทำงานประสานกันของคอมโพเนนท์ ดังนั้นขั้นตอนนี้จะทำการทดสอบการรวมคอมโพเนนท์ การโต้ตอบระหว่างคอมโพเนนท์

เป็นการทดสอบการยอมรับ นั่นคือ ทดสอบกับข้อมูลจริงของลูกค้า เพื่อตรวจสอบว่าได้ระบบตรงกับความต้องการของลูกค้าหรือไม่

ทดสอบจากหน่วยเล็กๆ ไปใหญ่

4. การวิวัฒนาการซอฟต์แวร์

- การเปลี่ยนแปลงในซอฟต์แวร์ สามารถเกิดขึ้นทั้งระหว่างการพัฒนาหรือหลังการพัฒนา (ค่าใช้จ่ายถูกกว่าการเปลี่ยนแปลงระบบฮาร์ดแวร์) ขึ้นอยู่กับความต้องการทางธุรกิจที่เปลี่ยนไป



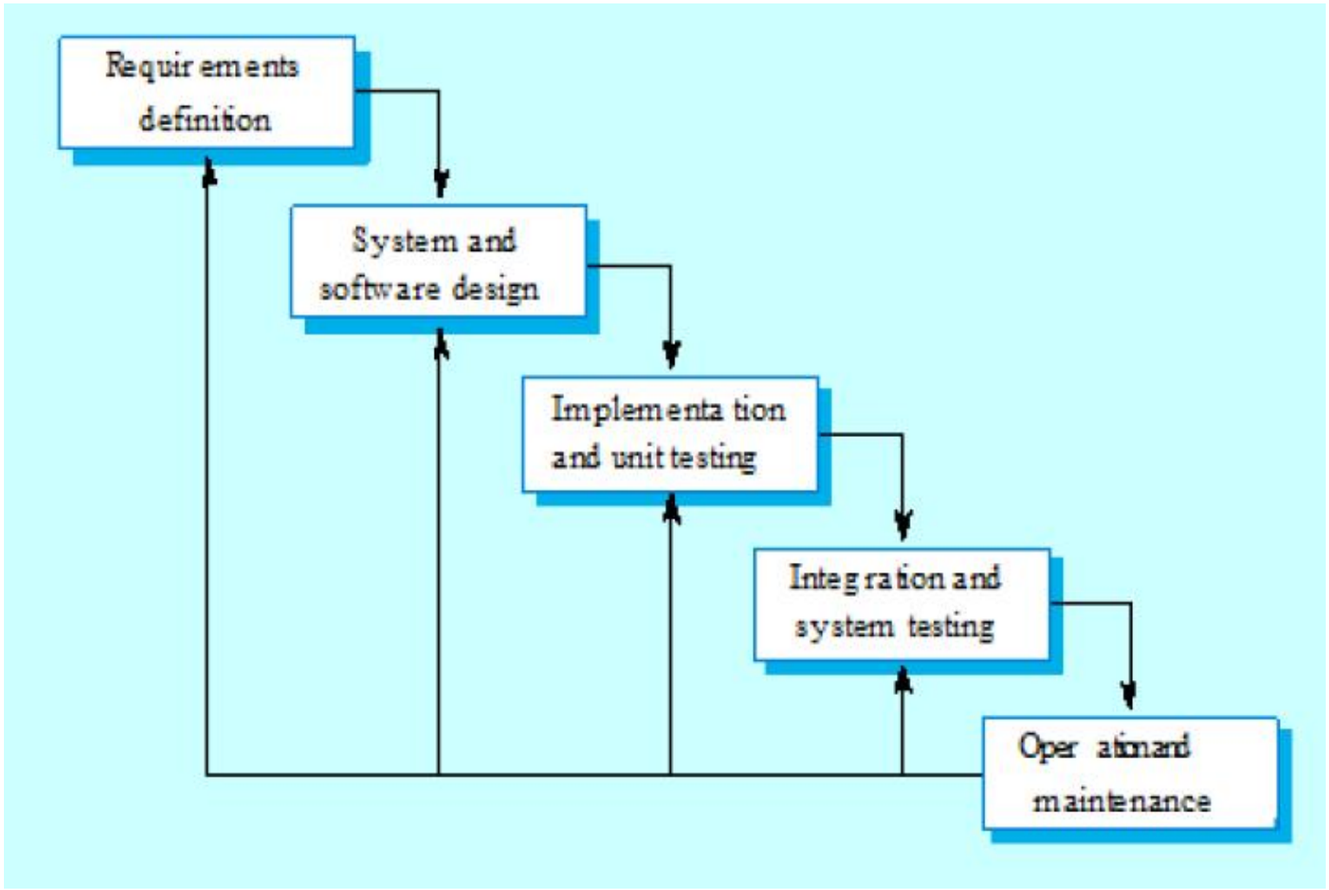
♦แบบจำลองกระบวนการซอฟต์แวร์ (Software process models)

- **แบบจำลองกระบวนการซอฟต์แวร์** หมายถึง การจำลองภาพของกระบวนการซอฟต์แวร์ เพื่อให้เห็นถึงการจัดโครงสร้างของกระบวนการ ในรูปแบบที่แตกต่างกันออกไป
- **รูปแบบทั่วไปของแบบจำลองกระบวนการซอฟต์แวร์**
  - แบบจำลองน้ำตก (Waterfall model)
  - แบบจำลองกระบวนการเชิงวิวัฒน์ (Evolutionary process)
- **รูปแบบอื่นๆ ของแบบจำลองกระบวนการซอฟต์แวร์**
  - แบบจำลองกระบวนการการพัฒนาโดยอาศัยองค์ประกอบ (Component-Based Development Process)
  - แบบจำลองแบบวิธีทางการ (Formal Methods Model)
  - การพัฒนาซอฟต์แวร์เชิงแง่มุม (Aspect-Oriented Software Development)

แบบจำลองน้ำตก (Waterfall model)

- **Waterfall model** เป็นแบบจำลองที่แบ่งกิจกรรมการดำเนินการพัฒนาซอฟต์แวร์เป็นลำดับขั้นแยกออกจากกันโดยเด็ดขาด โดยเรียงลำดับการทำงานก่อนหลัง
- **กิจกรรมใน Waterfall model** ประกอบด้วย
  - การวิเคราะห์ความต้องการและการกำหนดคุณลักษณะความต้องการ
  - การออกแบบระบบและซอฟต์แวร์
  - การสร้าง และทดสอบระดับย่อย
  - การทดสอบการรวมกันของหน่วยย่อย และการทดสอบทั้งระบบ
  - การใช้งานและบำรุงรักษา
- การดำเนินการ สามารถที่จะย้อนกลับเพื่อปรับปรุงซอฟต์แวร์ในรอบต่อไป

แบบจำลองน้ำตก (Waterfall model)



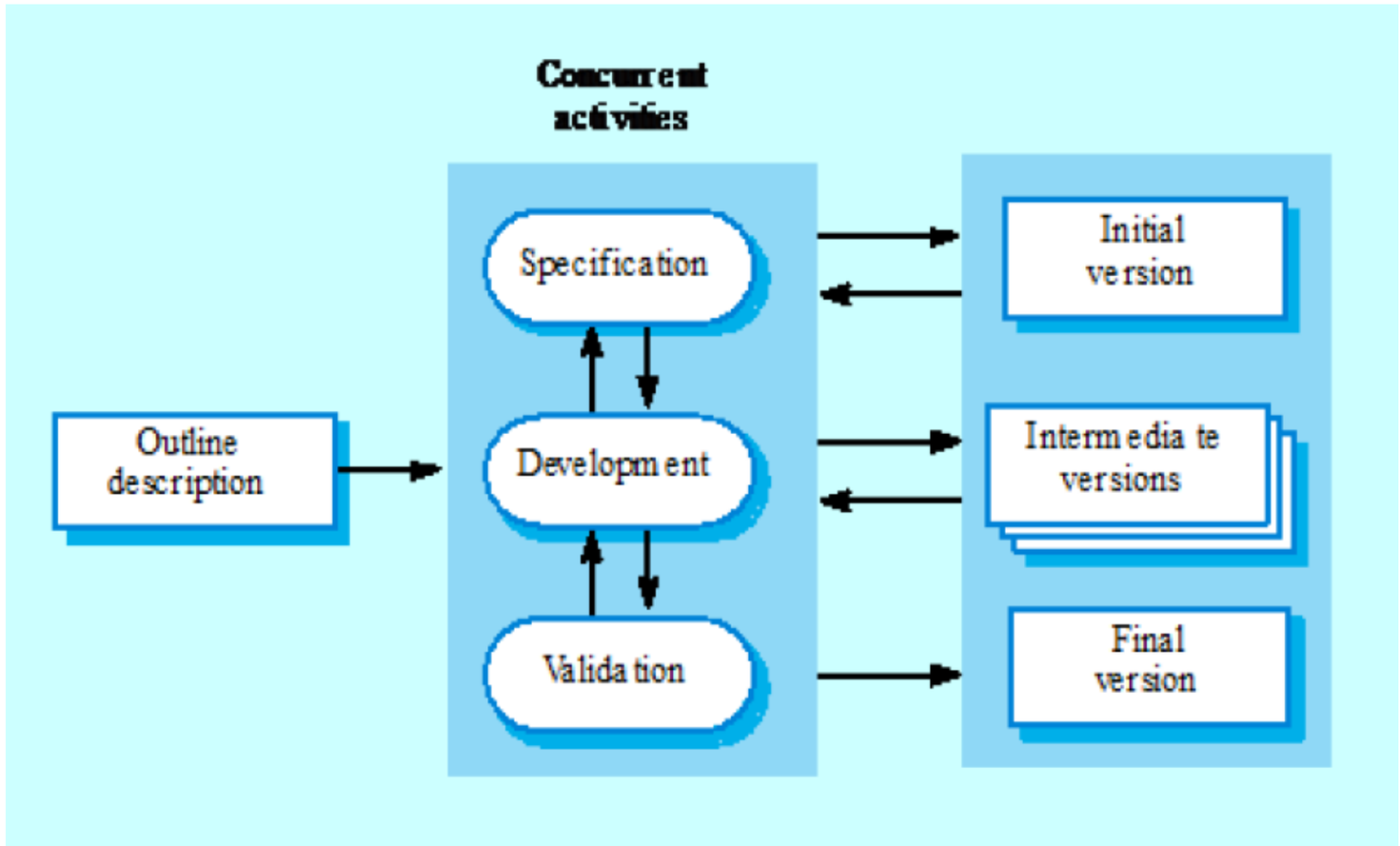
แบบจำลองน้ำตก (Waterfall model)

- **ปัญหาของ Waterfall model**
  - โครงการส่วนใหญ่มักจะไม่เรียงลำดับ มักจะมีการวนซ้ำ ดังนั้น เมื่อมีการเปลี่ยนแปลงใดๆ แบบจำลองน้ำตกอาจทำให้ทีมงานสับสนที่ทำตามขั้นตอนของแบบจำลองนี้
  - กรณีลูกค้ามีความต้องการที่ไม่ชัดเจน คลุมเครือ แบบจำลองนี้รองรับได้ยาก
  - ลูกค้าต้องรอซอฟต์แวร์จนกว่าจะเสร็จสิ้นกระบวนการขั้นตอนทั้งหมด ซึ่งหากมีข้อผิดพลาดขึ้น อาจทำให้เสียหายอย่างใหญ่หลวง
  - ทีมงานบางคนต้องรองานจากสมาชิกคนอื่นก่อนเริ่มงานที่ขึ้นแก่กันได้ ซึ่งเวลาที่ใช้ในการรออาจมากกว่าเวลาที่ใช้ทำงานได้

แบบจำลองกระบวนการเชิงวิวัฒน์ (Evolutionary process model)

- **แบบจำลองกระบวนการเชิงวิวัฒน์ (Evolution Process Model)** เป็นแบบจำลองที่มีการทำกิจกรรมในลักษณะวนซ้ำ โดยเมื่อลงมือพัฒนาแล้วจะมีการนำเสนอลูกค้าเพื่อนำคำแนะนำและติชมมาปรับปรุง และวนกลับมาแก้ใหม่ได้ในรอบต่อไป
- **กิจกรรมใน Evolution Process Model** (ที่ซ้อนทับกันอยู่) ประกอบด้วย
  - การกำหนดข้อกำหนดต่างๆ ของงาน (Specification)
  - การพัฒนา (Development)
  - การตรวจสอบความถูกต้อง (Validation)
- กิจกรรมเหล่านี้ไม่ได้แยกจากกันโดยเด็ดขาด มีการทำงานซ้อนทับกันอยู่โดยอาศัยข้อมูลย้อนกลับข้ามกิจกรรมได้

แบบจำลองกระบวนการเชิงวิวัฒน์ (Evolutionary process model)



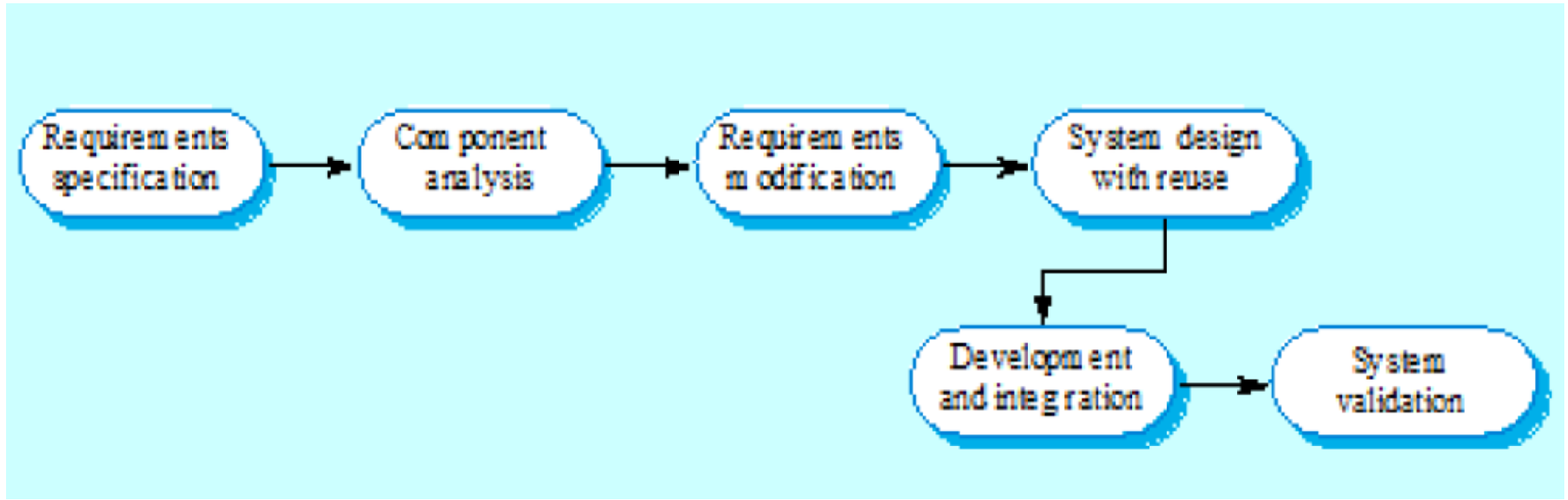
แบบจำลองกระบวนการเชิงวิวัฒน์ (Evolutionary process model)

- แนวทางการพัฒนาแบบวิวัฒนาการ :
  - **Exploratory development**
    - วัตถุประสงค์คือ การทำงานร่วมไปลูกค้า โดยเริ่มจากส่วนของระบบที่เป็นที่เข้าใจดีก่อน ทำการพัฒนา เพิ่มคุณลักษณะใหม่ตามที่ลูกค้านำเสนอเพิ่มเติม ทำซ้ำเช่นนี้จนกระทั่งส่งมอบระบบสุดท้าย
  - **Throwaway prototyping**
    - วัตถุประสงค์คือ ทำการทดลองกับความต้องการของลูกค้าที่เป็นที่เข้าใจน้อย โดยการสร้างโปรแกรมต้นแบบ (prototype) ขึ้นมา เพื่อดูว่าตรงกับความต้องการหรือไม่ ต้องเพิ่มเติม แก้ไข ปรับลดหรือไม่ หรือไม่ตรงกับวัตถุประสงค์เลย

กระบวนการพัฒนาโดยอาศัยองค์ประกอบ (Component-Based Development (CBD) Process

- **CBD หรือ CBSE** เป็นแนวทางการผลิตซอฟต์แวร์ที่อาศัยชิ้นส่วนที่ได้ถูกสร้างไว้แล้ว มาประกอบเป็นซอฟต์แวร์ใหม่ที่ต้องการ ด้วยหลักการของการนำกลับมาใช้ใหม่ (reusable) ทั้งในส่วนของโค๊ตและสถาปัตยกรรม
- **กิจกรรมใน CBD** ประกอบด้วย
  - การกำหนดคุณลักษณะความต้องการ
  - การวิเคราะห์ชิ้นส่วน (component) ต่างๆ
  - การปรับเปลี่ยนความต้องการ
  - การออกแบบระบบ ด้วยการนำของเดิมมาใช้ใหม่
  - การพัฒนาและการรวบรวม
  - การตรวจสอบความถูกต้องของระบบ
- วิธีการนี้กำลังเป็นที่นิยมมากขึ้นเพราะนำเอาชิ้นส่วนมาตรฐานมาใช้ใหม่

กระบวนการพัฒนาโดยอาศัยองค์ประกอบ (Component-Based Development (CBD) Process



แบบจำลองแบบวิธีทางการ (Formal Methods Model)

- **Formal Methods Model** เป็นการรวมเอาชุดกิจกรรมที่นำไปสู่ข้อกำหนดทางคณิตศาสตร์แบบเป็นทางการของซอฟต์แวร์ ทำให้ซอฟต์แวร์สามารถกำหนด พัฒนาและตรวจทางระบบคอมพิวเตอร์ได้โดยใช้สัญลักษณ์ทางคณิตศาสตร์ที่เข้มงวด

- **ข้อเด่นของวิธีทางการ คือ** ลดความคลุมเครือ ความไม่สมบรณ์และความไม่คงเส้นคงวา ข้อผิดพลาดจะถูกค้นพบและแก้ไขได้อย่างเป็นระบบ

- **ข้อเสียของวิธีทางการ คือ**
  - การพัฒนาแบบจำลองนี้ในปัจจุบันใช้เวลาและค่าใช้จ่ายมาก
  - นักพัฒนาซอฟต์แวร์ทั่วไปไม่มีความรู้พื้นฐานที่จะใช้งานวิธีนี้ จึงต้องเสียค่าใช้จ่ายในการอบรมมาก
  - มีความลำบากในการใช้แบบจำลองนี้เป็นกลไกในการสื่อสารระหว่างผู้พัฒนากับลูกค้า

แบบจำลองการพัฒนาซอฟต์แวร์เชิงแง่มุม (Aspect-Oriented Software Development)

- ระบบคอมพิวเตอร์สมัยใหม่ มีความซับซ้อนมากขึ้น ทำให้เกิดความ*กังวล*บางประการของลูกค้า อันได้แก่ ความมั่นคงปลอดภัย, ความล้มเหลวของระบบ, การประสานเวลาของงานย่อย, การจัดการหน่วยความจำ เป็นต้น

- ดังนั้นจึงเกิดการพัฒนาเชิงแง่มุม (Aspect-Oriented Software Development: AOSD) เพื่อพัฒนาแก้ไขในแง่มุมที่กังวลนั้น

- วิธีการนี้ยังไม่โตเต็มที่ในปัจจุบัน อย่างไรก็ตาม มีความเป็นไปได้สูงที่กระบวนการนี้จะรับเอาแบบจำลอง Spiral และแบบจำลองแบบทำไปพร้อมกันมาใช้ ทำคู่ขนานกันไป เนื่องจากมีความจำเป็นในการสร้างแง่มุม ซึ่งมีผลกระทบต่อหลายๆ ส่วนของซอฟต์แวร์

ประโยชน์และข้อจำกัดของแบบจำลองกระบวนการซอฟต์แวร์

แบบจำลอง	ประโยชน์	ข้อจำกัด
Waterfall model	- สามารถใช้งานได้ดีกับงานที่ต้องการความละเอียดถูกต้องสูง ที่มีการระบุและทำความต้องการเป็นอย่างดี - เหมาะสำหรับโครงการซอฟต์แวร์ขนาดใหญ่	ไม่สามารถรองรับความต้องการของลูกค้าที่เปลี่ยนแปลงอย่างรวดเร็ว เนื่องจากขาดความยืดหยุ่นในการแบ่งระยะของโครงการ ต้องให้ขั้นตอนหนึ่งเสร็จก่อนที่จะเริ่มอีกขั้นตอนหนึ่ง
Evolutionary Process	- ข้อกำหนดซอฟต์แวร์จะค่อยๆ ถูกพัฒนาเพิ่มเติม - สามารถรองรับความต้องการเพิ่มเติมจากลูกค้า - เหมาะสำหรับการพัฒนาโครงการซอฟต์แวร์ขนาดเล็กถึงปานกลาง	- หากระบบถูกพัฒนาอย่างรวดเร็ว การกำหนดค่าใช้จ่ายในการผลิตเอกสารสำหรับแต่ละเวอร์ชันจะไม่มีประสิทธิภาพ - ซอฟต์แวร์ที่ได้อาจมีโครงสร้างที่ไม่ดี - การบริหารจัดการ และการดูแลรักษา ทำได้ยาก
Component-based Development	- ช่วยลดต้นทุนในการพัฒนาลงได้มาก - ทำให้มั่นใจได้ว่าซอฟต์แวร์มีคุณภาพแน่นอน	คอมโพเนนท์ที่มีอยู่ในแหล่งเก็บ ต้องน่าเชื่อถือ มีคุณภาพ และต้องผ่านการทดสอบมาอย่างดี
Formal method	- ทำให้ซอฟต์แวร์ปราศจากข้อบกพร่อง เนื่องจากผ่านการพิสูจน์ทางคณิตศาสตร์	- ใช้เวลาและค่าใช้จ่ายมากในการพัฒนา ใช้ยาก
Aspect-Oriented SW Development	- ช่วยลดปัญหาในบางแง่มุมของระบบงาน	- การแก้ไขบางส่วน อาจมีผลกระทบกับหลายๆ ส่วนของซอฟต์แวร์

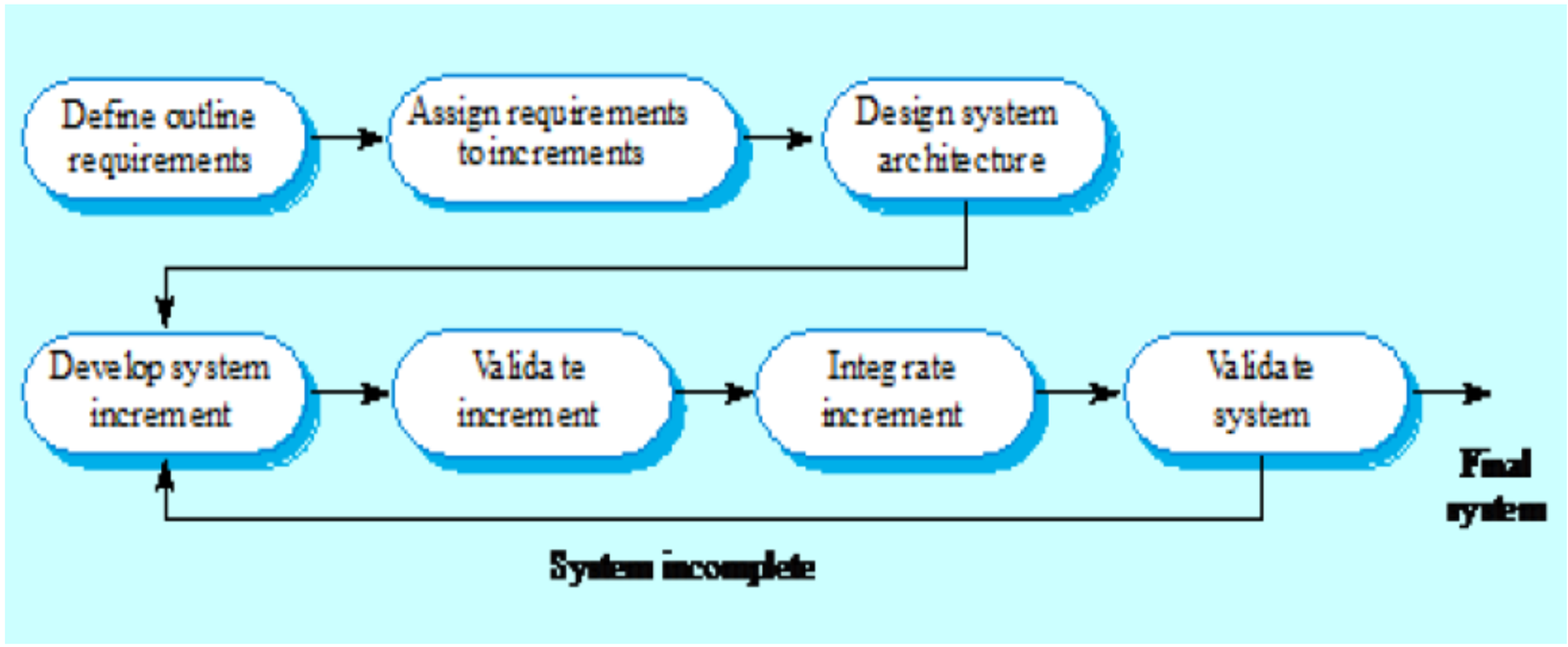
♦ การวนรอบของกระบวนการ (Process iteration)

- การระบุความต้องการของระบบนั้นมีอยู่คู่กับโครงการเสมอ ดังนั้น กระบวนการที่วนรอบคือการกลับไปแก้ไขงานในระยะแรกจะเป็นกระบวนการสำหรับระบบใหญ่เสมอ.
- การวนรอบนั้นใช้ได้กับแบบจำลองกระบวนการทั่วๆไป
- มีสองวิธีที่ใช้งานกันคือ
  - การส่งมอบแบบค่อยเพิ่มขึ้น (Incremental Model)
  - การพัฒนาแบบขดหอย (Spiral Development)

1. การส่งมอบแบบค่อยเพิ่มขึ้น (Incremental Model)

- วิธีนี้ การพัฒนาและการส่งมอบถูกแบ่งออกเป็นขั้นตอน แต่ละตอนจะมีการส่งมอบเพิ่มเติมตามที่กำหนดไว้ล่วงหน้า (ไม่ใช่ส่งมอบระบบทั้งหมดในครั้งเดียว)
- มีการจัดลำดับความสำคัญตามความต้องการของลูกค้า ความต้องการที่มีลำดับสำคัญสูงจะถูกส่งมอบก่อน
- ตัวอย่าง Word Processor
  - รุ่นแรก – ส่งมอบเพียงหน้าที่จัดการไฟล์พื้นฐาน, การตัดต่อ และการสร้างเอกสารพื้นฐาน
  - รุ่นที่สอง – เพิ่มหน้าที่การตัดต่อ และการสร้างเอกสารให้ดียิ่งขึ้น
  - รุ่นที่สาม – เพิ่มการตรวจสอบการสะกดคำ และตรวจสอบไวยากรณ์
  - รุ่นที่สี่ - เพิ่มความสามารถในการจัดเลย์เอาท์หน้าพิมพ์

1. การส่งมอบแบบค่อยเพิ่มขึ้น (Incremental Model)



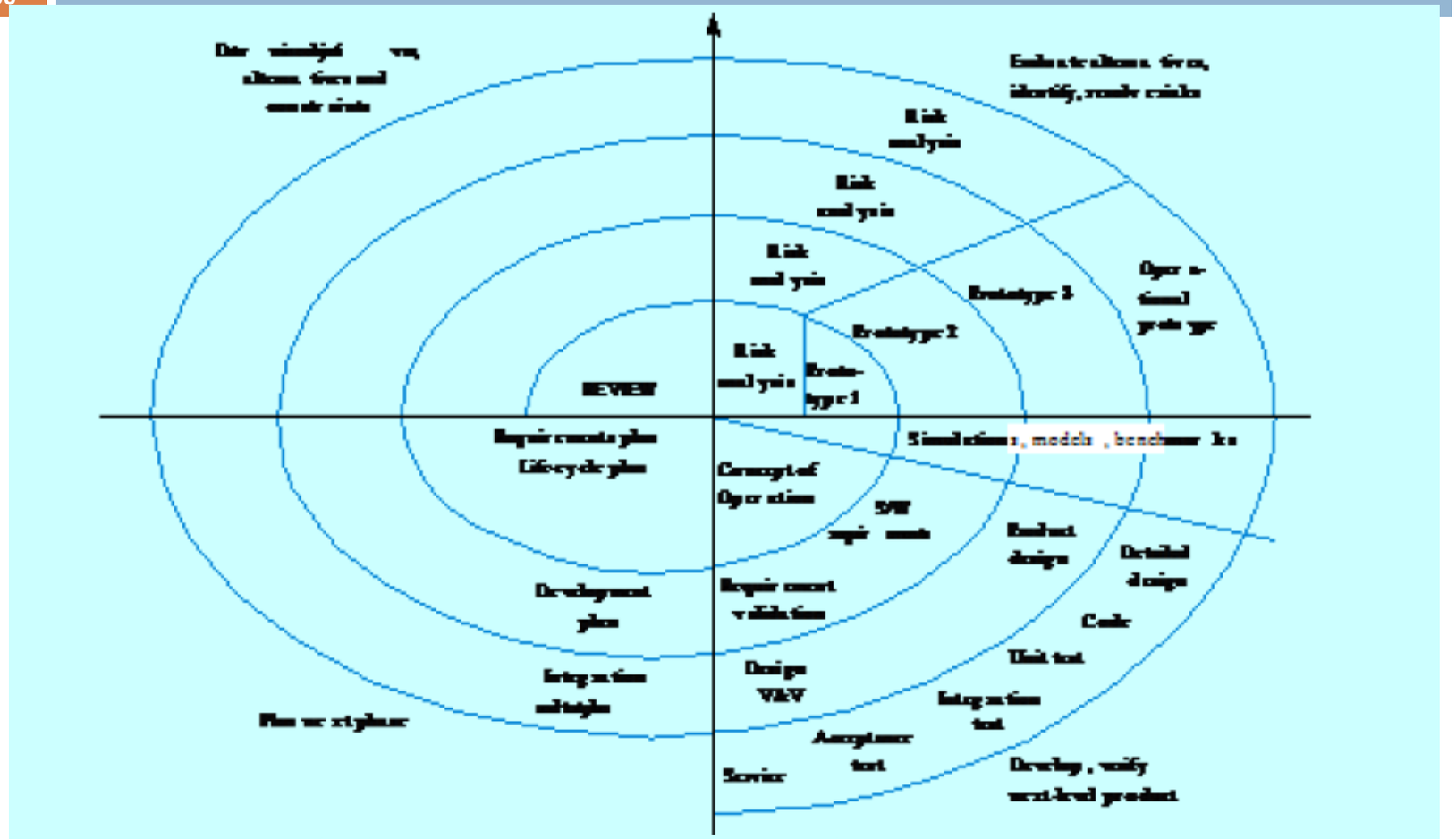
1. การส่งมอบแบบค่อยเพิ่มขึ้น (Incremental Model)

- การพัฒนาแอปพลิเคชันอย่างรวดเร็ว (Rapid Application Development –RAD) เป็นวิธีการหนึ่งของการพัฒนาแบบค่อยเพิ่มขึ้นที่เน้นวงจรพัฒนาสั้นๆ
- วิธี RAD นี้ได้ดัดแปลงแบบจำลองน้ำตกให้มีความเร็วสูง โดยแบ่งแอปพลิเคชันออกเป็นโมดูล แต่ละโมดูลทำงานคู่ขนานกันไปและต้องทำให้แล้วเสร็จภายในระยะเวลาอันรวดเร็ว ซึ่งอาจมีการใช้วิธีประกอบคอมโพเน้นท์ย่อยเข้าด้วยกัน
- ประโยชน์
  - ลูกค้าจะได้รับประโยชน์จากการส่งมอบงานในแต่ละ increment เร็วขึ้น
- ข้อเสีย
  - ต้องมีทรัพยากรบุคคลจำนวนมากเพื่อแบ่งเป็นทีมหลายๆ ทีม
  - โครงการอาจล้มเหลวได้ง่าย ถ้านักพัฒนาและลูกค้าไม่ทำตามกิจกรรมที่จำเป็น
  - ระบบที่ไม่สามารถแบ่งเป็นโมดูลได้ จะมีปัญหาในการสร้างคอมโพเนนท์ที่จะใช้
  - ไม่เหมาะกับระบบที่มีความเสี่ยงด้านเทคนิคสูง

2. การพัฒนาแบบขดหอย (Spiral development)

- รูปแบบของกระบวนการถูกนำเสนอในรูปแบบเกลียววน (spiral) แต่ละรอบในเกลียววนหมายถึงหนึ่งระยะ (phase) ของกระบวนการซอฟต์แวร์
- แต่ละรอบในเกลียววน (spiral) แบ่งออกเป็น 4 ส่วน
  - **การกำหนดวัตถุประสงค์ (Objective Setting)** → ระบุวัตถุประสงค์ให้กับ phase โครงการ
  - **การประเมินและลดความเสี่ยง (Risk assessment and reduction)** → ประเมินความเสี่ยงและเตรียมการลดความเสี่ยง
  - **การพัฒนาและการตรวจสอบ (Development and validation)** → เลือกโมเดลที่จะพัฒนาให้ระบบและทบทวนยืนยัน
  - **การวางแผน (Planning)** → ทบทวนแผนงานของโครงการ และวางแผนให้ระยะถัดไป
- วนรอบด้านในสุด อาจเป็นระยะ*การศึกษาความเป็นไปได้* วนรอบถัดมาเป็น*การกำหนดความต้องการ* รอบถัดมาเป็น*การออกแบบ* เป็นต้น

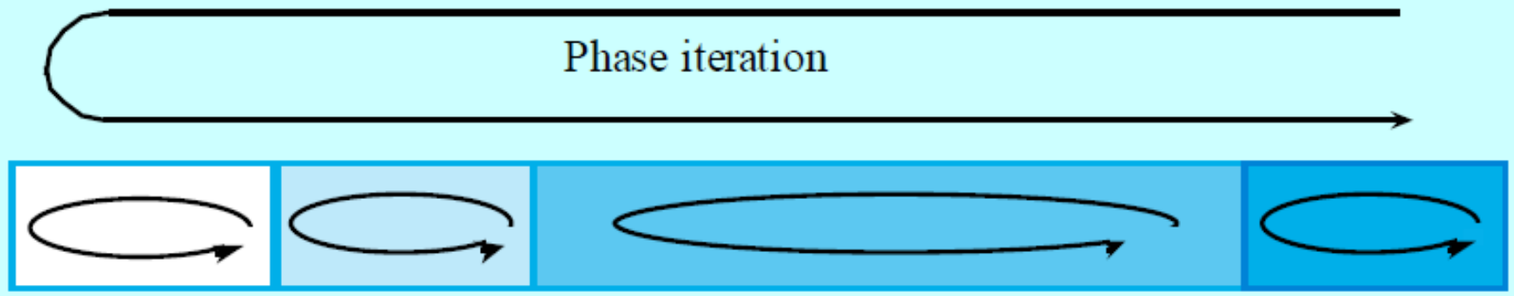
2. การพัฒนาแบบขดหอย (Spiral development)



♦ กระบวนการแบบรวมเป็นหนึ่งเดียว (RUP - Rational Unified Process)

- RUP เกิดขึ้นจาก Ivar Jacobson, Grady Booch, และ James Rumbaugh ซึ่งพยายามดึงเอาลักษณะที่ดีที่สุดของวิธีการวิเคราะห์และออกแบบเชิงวัตถุแบบดั้งเดิม ซึ่งเรียกว่า วิธีการยูนิฟายด์ (Unified Method) และรับเอาลักษณะเพิ่มเติมที่แนะนำโดยผู้เชี่ยวชาญด้านเชิงวัตถุ ผลที่ได้คือ UML (Unified Modeling Language)
- โดยปกติแล้ว RUP อธิบายได้ใน 3 มุมมอง
  - Dynamic - แสดงความก้าวหน้าของงานในช่วงเวลาต่างๆ
  - Static - แสดงกิจกรมของกระบวนการที่กำหนดเอาไว้
  - Practical – แนะนำแนวทางปฏิบัติในทางที่ดี ที่ถูกใช้ระหว่างกระบวนการ

ขั้นตอนต่างๆ ใน RUP



Inception	Elaboration	Construction	Transition
(การสื่อสารกับลูกค้า, กำหนดรูปแบบธุรกิจให้ระบบ)	(ทำความเข้าใจปัญหา, กำหนดกรอบงาน, วางแผนงาน,ประเมินความเสี่ยง, สร้างแบบจำลอง)	(เขียนโปรแกรม, จัดหาคอมโพเนนท์, ปรับการวิเคราะห์และออกแบบ, และทดสอบ)	(ส่งมอบระบบให้กับผู้ใช้งาน, ทดสอบการใช้งานได้ในสภาพแวดล้อมจริง, ทำคู่มือสนับสนุน)

หมายเหตุ:

- ขั้นตอนใน waterfall model มีกิจกรรมต่างๆ ของกระบวนการ ที่เท่ากัน
- ขั้นตอนใน RUP สัมพันธ์กับด้านธุรกิจอย่างใกล้ชิดมากกว่าด้านเทคนิค

ข้อดีของ RUP

- กำหนดการพัฒนาในแต่ละรอบตามความสำคัญของงาน
- บริหารความต้องการของลูกค้า
- ใช้การออกแบบโดยมีส่วนประกอบเป็นฐาน
- ใช้โมเดล UML แสดงรูปแบบของระบบ
- ตรวจทานดูแลให้ซอฟต์แวร์มีคุณภาพ
- ควบคุมการเปลี่ยนแปลงของซอฟต์แวร์

♦ เครื่องมือและวิธีการทางวิศวกรรมซอฟต์แวร์ (CASE Tools and Methodology in Software Engineering)

□ เครื่องมือทางวิศวกรรมซอฟต์แวร์ (CASE Tools)

- CASE tools หมายถึงซอฟแวร์ที่เป็นเครื่องมือที่มีส่วนประกอบช่วยสนับสนุนการทำงานในกิจกรรมต่างๆ ของงานวิศวกรรมซอฟแวร์ไม่ว่าจะเป็น ความต้องการ การออกแบบ การเขียนโปรแกรม และการทดสอบโปรแกรม
- CASE tools ถือเป็นเทคโนโลยีชนิดหนึ่ง ที่เพิ่มความสามารถให้กับซอฟแวร์จนกลายเป็นเครื่องมือช่วยแบ่งเบาภาระของนักพัฒนาระบบลงได้
- CASE tools สามารถจำแนกได้หลายประเภท โดยมีหลักการจำแนกหลายอย่าง โดยหากจำแนกตามกระบวนการทำงานขั้นตอนต่างๆ แล้ว สามารถแบ่ง ออกเป็น 8 กลุ่ม

เครื่องมือทางวิศวกรรมซอฟต์แวร์

1. เครื่องมือสำหรับการวิเคราะห์ (Software Requirement Tools)

- **<u>เครื่องมือในการสร้างแบบจำลองความต้องการ (Requirement Modeling Tools)</u>** เป็นเครื่องมือที่ใช้ในการดึงความต้องการ วิเคราะห์ กำหนด และตรวจสอบความต้องการด้านซอฟแวร์
- **<u>เครื่องมือการติดตามความต้องการ (Requirement Traceability Tools)</u>** เป็นเครื่องมือที่ใช้ติดตามความต้องการที่เปลี่ยนแปลงอยู่ตลอดเวลา

2. เครื่องมือออกแบบซอฟแวร์ (Software Design Tools) เป็นเครื่องมือที่ใช้สร้างและตรวจสอบการออกแบบซอฟแวร์ ปัจจุบันมีอยู่เป็นจำนวนมาก และส่วนใหญ่จะมีหน้าที่สนับสนุนการวิเคราะห์ความต้องการด้านซอฟแวร์ด้วย

3. เครื่องมือสร้างซอฟแวร์ (Software Construction Tools) เป็นเครื่องมือที่สนับสนุนการสร้างซอฟแวร์ ได้แก่ เครื่องมือแก้ไขโปรแกรม คอมไพเลอร์ อินเตอร์พรีเตอร์ ดีบักเกอร์

เครื่องมือทางวิศวกรรมซอฟต์แวร์

4. เครื่องมือทดสอบซอฟแวร์ (Software Testing Tools)

- **เครื่องสร้างกรณีทดสอบ (Testing Generation)** ใช้สร้างกรณีทดสอบซอฟแวร์
- **กรอบการปฏิบัติการทดสอบ (Test Execution Framework)** ใช้ทดสอบซอฟแวร์ภายใต้สภาพแวดล้อมที่มีการกำหนดไว้ล่วงหน้า
- **เครื่องมือประเมินผลการทดสอบ (Test Evaluation Tools)** ใช้สนับสนุนการประเมินผลการทดสอบ ว่าผลการทดสอบเป็นไปตามคาดหวังหรือไม่
- **เครื่องมือบริหารงานทดสอบ (Test Management Tools)** เป็นเครื่องมือสนับสนุนทุกกิจกรรมการทดสอบ
- **เครื่องมือวิเคราะห์ประสิทธิภาพการทดสอบ (Performance Analysis Tools)** เป็นเครื่องมือที่ใช้วัดผลและวิเคราะห์ประสิทธิภาพการทำงานของซอฟแวร์

เครื่องมือทางวิศวกรรมซอฟต์แวร์

5. เครื่องมือบำรุงรักษาซอฟแวร์ (Software Maintenance Tools) เป็นเครื่องมือที่ผู้ใช้บำรุงรักษาซอฟแวร์ที่มีอยู่แล้ว ให้คงสภาพที่ใช้การได้อย่างดี แบ่งเป็น 2 กลุ่มได้แก่

- เครื่องมือสร้างความเข้าใจ (Comprehension Tools) เป็นเครื่องมือที่ช่วยให้ทีมซ่อมบำรุงทำความเข้าใจกับโปรแกรมของซอฟแวร์ได้ง่ายขึ้น
- เครื่องมือรื้อปรับระบบใหม่ (Reengineering Tools) เป็นเครื่องมือที่ช่วยในกระบวนการรื้อโครงสร้างของซอฟแวร์ทีละส่วน เพื่อนำมาปรับหรือแก้ไขให้มีสภาพสมบูรณ์เหมือนเดิม

6. เครื่องมือจัดการโครงแบบ (Software Configuration Management Tools) เป็นเครื่องมือที่ใช้ติดตามการเปลี่ยนแปลงของทุกองค์ประกอบของซอฟแวร์ จัดการรุ่นของซอฟแวร์และวางจำหน่วยซอฟแวร์

เครื่องมือทางวิศวกรรมซอฟต์แวร์

7. เครื่องมือบริการงานวิศวกรรมซอฟแวร์ (Software Engineering Management Tools)

- เครื่องมือวางแผนและติดตามโครงการ (Project Planning and Tracking) ได้แก่ซอฟแวร์ที่ใช้ในการประมาณการแรงงาน และต้นทุน พร้อมทั้งจัดตารางงานด้วย
- เครื่องมือจัดการความเสี่ยง (Risk Management) ได้แก่ ซอฟแวร์ที่ระบุปัจจัยเสี่ยงประมาณการผลกระทบและติดตามความเสี่ยง
- เครื่องมือวัดผลโครงการ (Measurement) ได้แก่ ซอฟแวร์ที่ใช้ในการวัดผลทุกกิจกรรมของโครงการ

8. เครื่องมือคุณภาพซอฟแวร์ (Software Quality Tools) แบ่งเป็น 2 กลุ่ม ได้แก่

- เครื่องมือตรวจสอบคุณภาพ (Inspection Tools) ได้แก่ เครื่องมือที่ใช้ทบทวนและตรวจสอบคุณภาพของซอฟแวร์
- เครื่องมือวิเคราะห์คุณภาพ (Static Analysis Tools) ได้แก่เครื่องมือที่ใช้วิเคราะห์ลักษณะด้านต่างๆ ของซอฟแวร์

เครื่องมือทางวิศวกรรมซอฟต์แวร์

ประเด็นอื่นๆ นอกจากเครื่องมือที่กล่าวถึงข้างต้นแล้ว ยังมีประเด็นอื่นๆ เกี่ยวกับ CASE Tools ที่น่าสนใจ ได้แก่

- **Integrated CASE Environment** - เป็น CASE Tools ที่ประกอบด้วยฟังก์ชันการทำงานที่ครอบคลุมงานทุกด้านของสิศวกรรมซอฟแวร์ อีกทั้งยังสามารถประสานการทำงานเข้ากับ CASE Tool สำหรับขั้นตอนพื้นฐานของการพัฒนาซอฟแวร์อีกด้วย
- **Meta Tools** - เป็นเครื่องมือที่ใช้สร้างเครื่องมือ เช่น Editor ใช้สร้างโปรแกรมที่เป็นคอมไพเลอร์ เป็นต้น

ระเบียบวิธีทางวิศวกรรมซอฟต์แวร์

- **Heuristic Methodology** เป็นระเบียบวิธีที่ไม่มีแบบแผน (Informal Method) กล่าวคือ ไม่มีการนำวิธีการทางคณิตศาสตร์เข้าไปในขั้นตอนต่างๆ
  - Structured Methodology / Approach - มุ่งเน้นที่หน้าที่การทำงานของโปรแกรม
  - Object-Oriented Methodology - พิจารณาทั้งข้อมูลและหน้าที่การทำงานไปพร้อมๆ กัน
  - Data-Oriented Methodology - มุ่งเน้นที่ข้อมูลที่โปรแกรมจะต้องเข้าไปดำเนินการ
- **Formal Methodology** เป็นระเบียบวิธีที่อาศัยวิธีการทางคณิตศาสตร์เป็นพื้นฐาน
  - การระบุข้อกำหนดอย่างมีแบบแผน (Formal Specification) เป็นวิธีการอธิบายข้อกำหนดด้วยภาษาชนิดใดชนิดหนึ่ง ที่ได้มีการกำหนดนิยามของคำศัพท์และรูปแบบของไวยกรณ์ไว้อย่างเป็นทางการแล้ว
  - การทวนสอบอย่างมีแบบแผน (Formal Verification) เป็นวิธีการทวนสอบโดยใช้การพิสูจน์ทางตรรกะช่วยให้การทวนสอบซอฟแวร์มีประสิทธิภาพมากขึ้น