บทที่ 2
แนวคิดและทฤษฎีที่เกี่ยวข้อง

ในการวิจัยครั้งนี้ผู้วิจัยได้ศึกษาแนวคิด ทฤษฎี เอกสาร และงานวิจัยที่เกี่ยวข้อง ซึ่งจะนำเสนอสาระสำคัญโดยสรุปตามลำดับดังนี้

2.1 ไคลเอ็นต์ เซิร์ฟเวอร์ (Client/Server)

2.2 ระบบฐานข้อมูล (Database)

2.3 เว็บเบสแอปพลิเคชั่น (Web-Based Application)

2.4 หลักการวิศวกรรมซอฟแวร์ (Software Engineering)

2.1 ไคลเอ็นต์ เซิร์ฟเวอร์ (Client / Server)

เป็นระบบการทำงานแบบ Distributed Processing หรือการประมวลผลแบบกระจาย โดยจะแบ่งกันประมวลผลระหว่างเครื่องเซิร์ฟเวอร์กับเครื่องเวิร์กสเตชั่น แทนที่โปรแกรม Application จะวิ่งทำงานอยู่เฉพาะเครื่องเซิร์ฟเวอร์ ก็จะแบ่งการคำนวณของโปรแกรม Application มาทำงานบนเครื่องเวิร์กสเตชั่นด้วยและเมื่อใดที่เครื่องเวิร์กเตชั่นต้องการผลลัพธ์ของข้อมูลบางส่วน จะมีการเรียกใช้ไปยังเครื่องเซิร์ฟเวอร์เพื่อให้เซิร์ฟเวอร์นำเฉพาะข้อมูลบางส่วนเท่านั้นส่งกลับมาให้เครื่องเวิร์กสเตชั่นเพื่อทำการคำนวณข้อมูลนั้นต่อไป

2.1.1 รูปแบบของระบบไคลเอ็นต์/เซิร์ฟเวอร์ ที่ใช้งานจะมีอยู่ 4 ชนิด

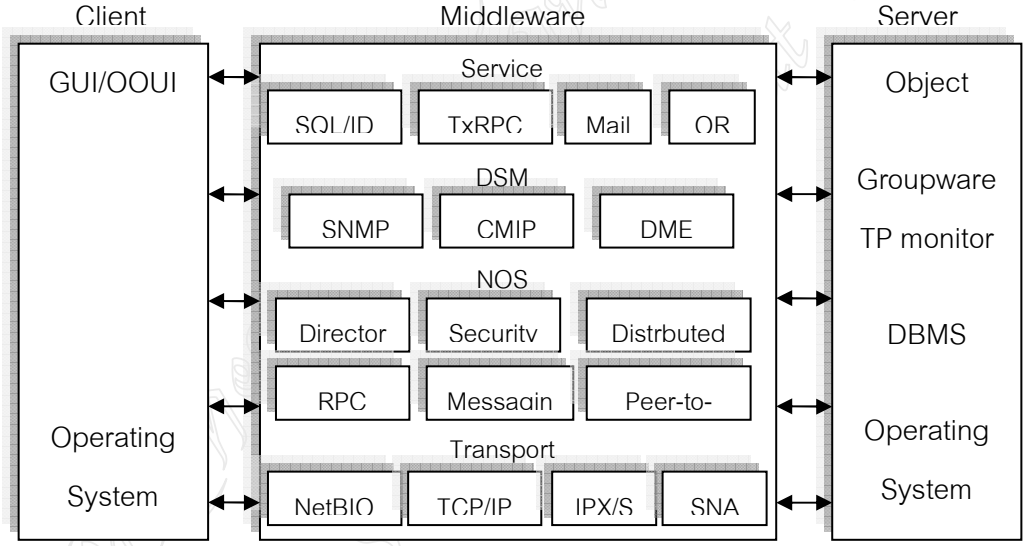
2.1.1.1 Stand alone Client/Server การทำงานแบบนี้ผู้ให้บริการหรือเซิร์ฟเวอร์จะอยู่บนเครื่องเดียวกันผู้ขอใช้บริการหรือไคลเอ็นต์ ทำให้มีความเก็นในการติดต่อสื่อสารระหว่างผู้ให้บริการและผู้ขอใช้บริการสูงมาก แต่ประสิทธิภาพในการประมวลผลระบบฐานข้อมูลจะลดลงบ้าง ระบบนี้เรียกอีกอย่างว่า Tiny client/server

2.1.1.2 Department Client/Server หรือ LAN based single server การทำงานแบบนี้จะมีผู้ให้บริการเกี่ยวกับฐานข้อมูล แอพพลิเคชั่น ฯลฯ อยู่บนเครื่องเซิร์ฟเวอร์ และผู้ขอใช้บริการทั้นหลายจะอยู่บนเครื่องไคลเอ็นต์ โดยจะเชื่อมต่อกันด้วยระบบเครือข่ายท้องถิ่น (LAN) และมิดเดิลแวร์ (Middleware) เป็นตัวกลางที่ทำงานอยู่ระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ การติดต่อสื่อสารกันระหว่างผู้ให้บริการ และผู้ขอบริการจะช้ากว่าแบบ stand alone เพราะจะต้องติดต่อผ่านระบบเครือข่ายยิ่งถ้ามีผู้ขอบริการเข้ามาดึง

ข้อมูลกันครั้งละมากๆ หลายๆ ครั้ง เครื่อง ประสิทธิภาพจะลดลงอย่างเห็นได้ชัด วิธีเพิ่มประสิทธิภาพก็คือการเพิ่มเครื่องเซิร์ฟเวอร์ขึ้นในระบบ

2.1.1.3 Workgroups Client/Server การทำงานแบบเวิร์กกรุ๊ปนี้จะเป็นกลุ่มของเซิร์ฟเวอร์ที่หลากหลายแพลตฟอร์ม หลายผู้ผลิต มีความแตกต่างกันของเซิร์ฟเวอร์ แต่ทั้งหมดนี้จะเชื่อมต่อกันทางระบบเครือข่าย LAN และ WAN และใช้มิดเดิลแวร์มาตรฐานในการทำงาน

2.1.1.4 Enterprise Client/Server การทำงานแบบเอ็นเทอร์ไพท์หรือระดับองค์กรจะทำให้มีการเชื่อมโยงเครื่องเซิร์ฟเวอร์หรือโฮสต์ต่างแพลตฟอร์มเข้าด้วยกันทำให้มีการใช้ทรัพยากรบนระบบได้อย่างมีประสิทธิภาพสูงสุด โดยที่ไคลเอ็นต์สามารถจะเลือกใช้ทรัพยากร ฐานข้อมูลจากเซิร์ฟเวอร์เครื่องใดก็ได้ผ่านทางมิดเดิลแวร์



รูปที่ 2.1 โครงสร้างของไคลเอ็นต์/เซิร์ฟเวอร์

- **Client** เป็นส่วนที่จะรันแอพพลิเคชั่นบนไคลเอ็นต์ โดยใช้ระบบ GUI (Graphical User Interface) หรือ OOUI (Object Oriected User Interface) หรือ DSM (Distributed System Management) เป็นการติดต่อกับ User ผ่านระบบกราฟฟิกส์ซึ่งทำงานแบบเชิงวัตถุ (Object)

- **Middleware** เป็นส่วนที่ทำงานอยู่ระหว่างไคลเอ็นต์และเซิร์ฟเวอร์เป็นเสมือนสะพานเชื่อมการทำงาน สามารถแบ่งออกเป็น 4 แบบคือ Service Specific , DSM , NOS และ Transport stack

- **Server** เป็นส่วนที่จะรันแอพพลิเคชั่นในการจัดการทรัพยากรต่างๆ สำหรับระบบไคลเอ็นต์/เซิร์ฟเวอร์ สามารถแบ่งออกได้ 4 แบบด้วยกันคือ

  - ระบบฐานข้อมูล SQL (DBMS)

- ระบบจัดการทรานส์แอคชั่น (TP monitor)
- ระบบกรุ๊ปแวร์ (Groupware)
- ระบบอ็อบเจ็กต์แบบกระจาย (Distributed objects)

2.2 ระบบฐานข้อมูล (Database System)

เป็นกลุ่มของข้อมูลที่ถูกเก็บรวบรวมไว้ โดยมีความสัมพันธ์ซึ่งกันและกัน เพื่อลดความซ้ำซ้อนของข้อมูลและเก็บข้อมูลเหล่านี้ไว้ที่ศูนย์กลาง เพื่อที่จะนำข้อมูลเหล่านี้มาใช้ร่วมกัน

การจัดการฐานข้อมูล (Database Management) คือ การบริหารแหล่งข้อมูลที่ถูกเก็บรวบรวมไว้ที่ศูนย์กลาง เพื่อตอบสนองต่อการใช้งานอย่างมีประสิทธิภาพและลดการซ้ำซ้อนของข้อมูล รวมทั้งลดความขัดแย้งของข้อมูลที่เกิดขึ้นภายในองค์กรด้วย

การจัดการฐานข้อมูลต้องอาศัยโปรแกรมที่ทำหน้าที่ในการกำหนดลักษณะข้อมูลที่จะเก็บไว้ในฐานข้อมูล อำนวยความสะดวกในการบันทึกข้อมูลลงในฐานข้อมูล กำหนดผู้ที่ได้รับอนุญาตให้ใช้ฐานข้อมูลได้ พร้อมกับกำหนดด้วยว่าให้ใช้ได้แบบใด เช่น ให้อ่านข้อมูลได้อย่างเดียวหรือให้แก้ไขข้อมูลได้ด้วย นอกจากนั้นยังอำนวยความสะดวกในการค้นหาข้อมูล การแก้ไขปรับปรุงข้อมูล ตลอดจนการจัดทำข้อมูลสำรองด้วย โดยอาศัยโปรแกรมที่เรียกว่า ระบบการจัดการฐานข้อมูล(Database Management System: DBMS) ซึ่งโปรแกรมที่ได้รับความนิยมในการจัดการฐานข้อมูล ได้แก่ Microsoft Access, Oracle, Informix, dBase, FoxPro, และ Paradox เป็นต้น

2.2.1 ส่วนประกอบของตารางข้อมูลในฐานข้อมูล

โดยทั่วไปแล้วตารางข้อมูลที่ใช้งานกันจะประกอบด้วย แถว (Row) และคอลัมน์ (Column) ต่างๆ แต่ถ้ามองกันในรูปแบบของฐานข้อมูลแล้ว เราจะเรียกรายละเอียดในแถวว่า เรคคอร์ด (Record) และเรียกรายละเอียดในแนวคอลัมน์ว่า ฟิลด์ (Field) ในฐานข้อมูล 1 ระบบ อาจประกอบด้วยตารางข้อมูลมากกว่า 1 ตาราง ฐานข้อมูลที่มีตารางข้อมูลมากกว่า 1 ตาราง และมีตารางตั้งแต่ 1 คู่ขึ้นไปที่มีความสัมพันธ์กันด้วยฟิลด์ใดฟิลด์หนึ่ง เราเรียกฐานข้อมูลประเภทนี้ว่า “ฐานข้อมูลเชิงสัมพันธ์” หรือ Relational Database

2.2.2 โครงสร้างของฐานข้อมูลประกอบด้วย

2.2.2.1 Character คือ ตัวอักขระแต่ละตัว / ตัวเลข / เครื่องหมาย

2.2.2.2 Field คือ เขตข้อมูล / ชุดข้อมูลที่ใช้แทนความหมายของสื่อโครงสร้าง เช่น ชื่อของบุคคล ชื่อของวัสดุสิ่งของ

2.2.2.3 Record คือ ระเบียน หรือรายการข้อมูล เช่น ระเบียนของพนักงานแต่ละคน

2.2.2.4 Table /File คือ ตาราง หรือแฟ้มข้อมูล ประกอบขึ้นด้วยระเบียนต่างๆ เช่น ตารางข้อมูลของบุคคล ตารางข้อมูลของวัสดุสิ่งของ

2.2.2.5 Database คือ ฐานข้อมูล ประกอบด้วยตาราง และแฟ้มข้อมูลต่างๆ ที่เกี่ยวข้อง หรือมีความสัมพันธ์กัน

2.3 เว็บเบสแอปพลิเคชั่น (Web-Based Application)

ระบบงานที่ถูกพัฒนาขึ้นใช้งานบนบราวเซอร์ผ่านระบบเครือข่าย ซึ่งทำงานได้ทั้งบนอินเตอร์เน็ต

2.3.1 ข้อดีของเว็บเบสแอพพลิเคชั่น

2.3.1.1 ข้อมูลต่าง ๆ ในระบบมีการไหลเวียนในแบบ Online ทั้งแบบ Local (ภายในวง LAN) และ Global (ออกไปยังเครือข่ายอินเตอร์เน็ต) ทำให้เหมาะสำหรับงานที่ต้องการข้อมูลแบบ Real Time

2.3.1.2 ระบบมีประสิทธิภาพ แต่ใช้งานง่าย เหมือนกับท่านทำกำลังท่องเว็บ

2.3.1.3 ระบบงานที่พัฒนาขึ้นมาจะตรงกับความต้องการกับหน่วยงาน หรือห้างร้านมากที่สุด ไม่เหมือนกับโปรแกรมสำเร็จรูปทั่วไป ที่มักจะจัดทำระบบในแบบกว้างๆ ซึ่งมักจะไม่ตรงกับความต้องการที่แท้จริง

2.3.1.4 ระบบสามารถโต้ตอบกับลูกค้า หรือผู้ใช้บริการแบบ Real Time ทำให้เกิดความประทับใจ เครื่องที่ใช้งานไม่จำเป็นต้องติดตั้งโปรแกรมใดๆ เพิ่มเติมทั้งสิ้น

2.3.2 ความรู้เกี่ยวกับภาษาพีเอชพี (PHP)

Rasmus Lerdorf ผู้สร้างภาษา PHP ได้เริ่มจากการเขียนสคริปต์ Perl CGI ใส่ไว้ในโฮมเพจประวัติส่วนตัว เพื่อบันทึกข้อมูลผู้ที่เข้าเยี่ยมชมโฮมเพจ แต่เนื่องจาก Lerdrf เห็นว่าการเขียน CGI ด้วย Perl นั้นออกจะเยิ่นเย้อเกินไป จึงได้ตัดสินใจเขียนโปรแกรมขึ้นใหม่ด้วยภาษา C ที่สามารถแยกส่วนที่เป็นภาษา HTML ออกจากส่วนที่เป็นภาษา C เพื่อแยกประมวลผลแล้วทำการสร้างโค้ด HTML ขึ้นมาใหม่ โดยตั้งชื่อโปรแกรมนี้ว่า Personal Home Page Tools (PHP-Tools) และได้เริ่มแจกจ่ายโค้ดออกไปในลักษณะฟรีแวร์ (ซึ่งในขณะนั้น Open source ยังไม่เป็นที่รู้จักกันมากนัก) ต่อมาจึงได้เริ่มเปิดให้ผู้ที่สนใจเข้าร่วมปรับปรุงและพัฒนา จนพัฒนาเป็น PHP/FI ที่เริ่มเป็นที่นิยมมากขึ้น

จนกระทั้ง Zeev Suraski และ Andi Gutmans ได้ร่วมกันเขียนโค้ดขึ้นใหม่โดยได้มีการปรับปรุงให้ดีขึ้นเป็นอย่างมากในหลายๆด้าน ทั้งด้านประสิทธิภาพ การสนับสนุนการโปรแกรมเชิงวัตถุ และในด้านอื่นๆ อีกมากมายหลายประการจนเกิดเป็น PHP 3 ซึ่งเป็นเวอร์ชั่นที่ได้รับความนิยมเป็นอย่างมาก

แต่เมื่อมีผู้ใช้เป็นจำนวนมาก จึงมีการนำไปใช้ในงานที่ซับซ้อนขึ้น ด้วยเหตุนี้ Zeev Suraski และ Andi Gutmans ผู้พัฒนา PHP 3 จึงตัดสินใจเขียนโค้ดขึ้นใหม่ทั้งหมด และได้ตั้งชื่อว่า Zend engine (มาจาก Zeev และ Andi) ซึ่งเป็นหัวใจของ PHP 4 ในปัจจุบัน ส่วน PHP 5 เป็นเวอร์ชันที่จัดได้ว่าเป็นการ

พลิกโฉมการโปรแกรมเชิงวัตถุด้วย PHP เนื่องจากมีการเปลี่ยนแปลงไปสู่การโปรแกรมเชิงวัตถุที่สมบูรณ์แบบมากยิ่งขึ้น

PHP เป็นภาษาสคริปต์แบบเซิร์ฟเวอร์ไซด์ (server-side scripting language) หมายถึงการประมวลผลจะเกิดขึ้นบนเครื่องแม่ข่าย หรือเซิร์ฟเวอร์ (server) แล้วจึงสร้างผลลัพธ์เป็นภาษา HTML ส่งมาให้กับเครื่องลูกข่าย หรือไคลเอนท์ (client) เพื่อแสดงผล ซึ่งลดภาระการส่งข้อมูลจำนวนมากเพื่อมาประมวลผลบนเครื่องลูกข่าย

ภาษา PHP เป็นโอเพ่นซอร์ส (Open Source) ซึ่งสามารถดาวน์โหลด PHP (พร้อม source code) มาใช้งานได้ฟรีจากเว็ปไซต์ของ PHP (www.php.net/download.php) ส่วนคู่มือการใช้งาน (PHP Manual) นั้นสามารถเรียกดูได้จาก www.php.net/docs.php ซึ่งสามารถเรียกดูในแบบออนไลน์ได้ทันที (HTML) หรือหากต้องการดาวน์โหลดก็มีให้เลือกทั้งในรูปแบบเว็บเพจ (HTML) และไฟล์ช่วยเหลือในแบบของ Windows (.chm) โดยสามารถดาวน์โหลดได้ที่ www.php.net/download-docs.php

2.3.3 MySQL

เป็นโปรแกรมฐานข้อมูล มีหน้าที่เก็บข้อมูลอย่างมีโครงสร้าง และรองรับคำสั่ง SQL เป็นเครื่องมือสำหรับเก็บข้อมูลอย่างมืออาชีพ ยังมีเครื่องมืออีกหลายอย่าง ที่ท่านต้องใช้ร่วมกันอย่างสอดคล้อง จึงจะนำไปพัฒนาระบบฐานข้อมูลซับซ้อน ตามความต้องการของผู้ใช้ได้สำเร็จสมประสงค์ เช่น การบริการเว็บ ภาษาสำหรับพัฒนาเว็บ ระบบปฏิบัติการ และคอมพิวเตอร์ที่เหมาะสม การใช้ MySQL ในฐานะนักเรียน เป็นเพียงจุดเริ่มต้นของการสร้างระบบที่สมบูรณ์ แม้นักเรียนจะพัฒนาระบบฐานข้อมูลเป็นโครงงานก่อนจบได้สมบูรณ์ แต่นั่นก็เป็นเพียงระบบหนึ่ง การหาเวลาศึกษาหลายๆ ระบบจะทำให้นักเรียนเข้าใจระบบฐานข้อมูลมากขึ้น

ข้อดีของ MySQL

1. MySQL มีขนาดเล็กและใช้ทรัพยากรน้อย
2. เป็นฐานข้อมูลที่ได้รับความนิยมมากที่สุดในการเขียนแอพพลิเคชัน PHP
3. มีเครื่องมือมากมายในการจัดการ ทั้งแบบที่เป็นกราฟฟิกและเว็บ
4. สามารถติดตั้งบนวินโดวส์เช่นเดียวกันกับ Linux/FreeBSD

2.4 **หลักการวิศวกรรมซอฟแวร์** (Software Engineering)

กระบวนการในการออกแบบและพัฒนาซอฟต์แวร์ด้วยวิธีการทางวิศวกรรมนี้ก็ยังคงมีลักษณะเป็นการศึกษาหาวิธีดำเนินงานที่เหมาะสมที่สุด เพราะเท่าที่ได้พัฒนาขั้นตอนการดำเนินการแบบต่างๆ ตลอด

มา ปรากฏว่าเราไม่อาจหาข้อสรุปรวมได้ว่า วิธรการใดเป็นวิธรการที่ดีที่สุด บางวิธีอาจจะดีกว่าวิธีการหนึ่งในสถานการณ์ที่แตกต่างกันออกไป ดังนั้นจึงไม่อาจมีคำตอบที่แน่นอนสำหรับทุกสถานการณ์ได้

2.4.1 Software Process มีอยู่ 4 ขั้นตอนใหญ่ๆ ดังนี้

2.4.1.1 Specification คือ ขั้นตอนการกำหนดคุณสมบัติและความต้องการของระบบ โดยมีขั้นตอนที่เกี่ยวข้องกับความต้องการ (Requirement) ดังนี้

- ศึกษาความเป็นไปได้ เป็นสิ่งแรกที่ควรจะทำในการพัฒนาระบบหรือซอฟต์แวร์ ใดๆ ซึ่งจะเป็นการศึกษาว่าเป็นไปได้หรือไม่ที่จะพัฒนาหรือพัฒนาและจะพบกับปัญหาอะไรมากน้อยแค่ไหน
- เก็บรวบรวมและวิเคราะห์ความต้องการ เกิดจากการเลือกใช้เครื่องมือหรือเทคนิคในการเก็บรวบรวมข้อมูล ซึ่งเทคนิคหนึ่งคือการสัมภาษณ์ผู้ใช้หรือผู้มีส่วนได้ส่วนเสียกับการพัฒนาระบบซอฟต์แวร์ และนำมาวิเคราะห์ข้อมูล ผลลัพธ์ที่ได้คือ โมเดลระบบ (System Model)
- กำหนด Specification ซึ่งเป็นกระบวนการกำหนด Specification และอาจมีการเก็บ Requirement เพิ่มเติมในกรณีที่ไม่ถูกต้องหรือไม่เพียงพอ ทำให้ได้ User Requirement และ System Requirement ซึ่งมีลักษณะการอธิบายเป็นหัวข้ออย่างชัดเจนว่าต้องการให้ระบบทำอะไรบ้าง
- ตรวจสอบ Requirement ว่าถูกต้อง ไม่คลุมเครือ และนำไปให้ผู้ใช้หรือผู้มีส่วนได้ส่วนเสียกับการพัฒนาระบบซอฟต์แวร์อ่านและตรวจสอบว่าตรงตามความต้องการหรือไม่ ซึ่งถ้าตรงความต้องการ จะนำไปเขียนเป็นเอกสารความต้องการ (Requirement Document)

2.4.1.2. Design คือ ขั้นตอนการออกแบบระบบ หรือ ออกแบบตัว Software ซึ่งมีกระบวนการในการออกแบบระบบ (System Design) ที่สามารถนำขั้นตอนของ Requirement Specification มาเขียนและออกแบบได้ ดังต่อไปนี้

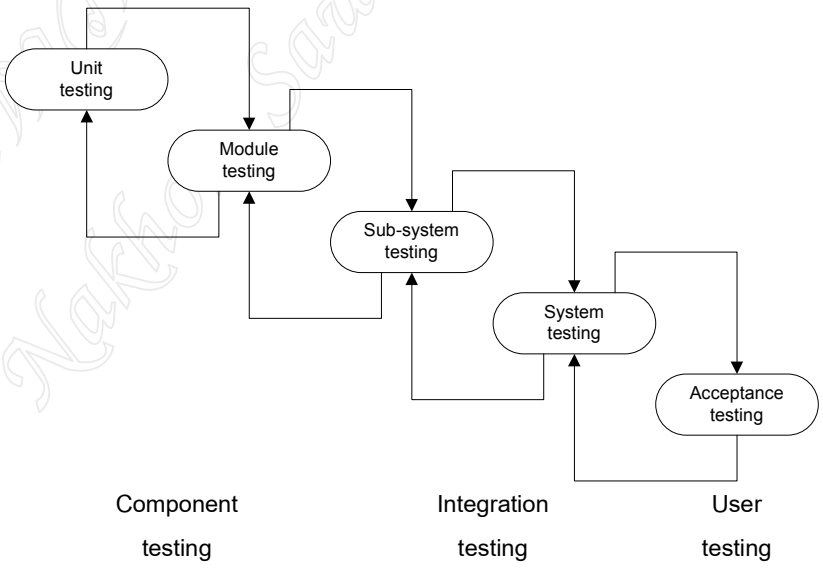
- การออกแบบสถาปัตยกรรมระบบ (System architecture) หรือมักจะเรียกเป็น Block Diagram ซึ่งจะบอกว่าระบบประกอบด้วยส่วนสำคัญอะไรบ้าง มีการเชื่อมโยงกันอย่างไร ต่อมาก็เขียน Software Specification (หรือ Functional Specification) ซึ่งบอกว่าซอฟต์แวร์สามารถทำอะไรได้บ้าง
- การออกแบบ Interface ได้เป็น Interface Specification ซึ่งจะเป็นการแสดงรายละเอียดของ Interface โดยเฉพาะ เช่น ลักษณะหน้าจอเป็นอย่างไร ใช้อุปกรณ์อะไรเป็น Input / Output ตรงส่วนไหนใช้เมาส์ คีย์บอร์ด หรือกดปุ่มไหนทำอะไรบ้าง

- การแบ่งระบบทั้งหมดออกเป็นระบบย่อยๆ (Component Specification) ว่าส่วนไหนทำอะไรบ้าง มีฟังก์ชั่นอะไร
- การออกแบบและจัดการฐานข้อมูล (Database System) จะวิเคราะห์ส่วนของข้อมูลว่าควรจะมีการจัดเก็บอย่างไรในรูปแบบอะไรและใช้ตัวเชื่อมโยงอะไร
- การออกแบบระบบอัลกอริทึม ซึ่งคือลำดับขั้นตอนของความคิดในการแก้ปัญหาการทำงานของโปรแกรม

2.4.1.3. Validation คือ รวมตั้งแต่การพัฒนาขึ้นมาและทดสอบว่ามันใช้งานได้จริงตามต้องการหรือไม่ โดยมีขั้นตอนในการออกแบบระบบการตรวจสอบ (Validation) ที่แบ่งออกเป็น 3 ส่วนใหญ่ๆ ดังนี้

- ทดสอบแต่ละส่วนประกอบของระบบ (Component testing)
- ทดสอบการนำส่วนประกอบของระบบมารวมเข้าด้วยกัน (Integration testing)
- ทดสอบใช้งานจริงโดยผู้ใช้ (User testing)

การเลือกว่าจะใช้วิธีไหนในการทดสอบขึ้นอยู่กับความเหมาะสมของขนาดของ Software และความต้องการระดับของความละเอียดของโปรแกรม



รูปที่ 2.2 ขั้นตอนการตรวจสอบ

Unit testing: เป็นการทดสอบว่ามีการกำหนดฟังก์ชั่นในการเรียกใช้อะไรบ้าง โดยบอกได้ว่ารับ input อะไรเข้ามา แล้วได้ output อะไรออกไปบ้าง

Module testing: เป็นการทดสอบความถูกต้องของฟังก์ชันการทำงานในแต่ละส่วนย่อยๆ ว่าทำงานได้ถูกต้องและมีประสิทธิภาพเพียงพอหรือไม่

Sub-system testing: เป็นการทดสอบระบบย่อยๆ ก่อนนำเข้ามารวมเป็นระบบใหญ่ เช่น ระบบบัญชี ระบบการเงิน ระบบบุคลากร

System testing: เป็นการทดสอบระบบใหญ่หรือระบบที่เป็นโครงการทั้งหมดภายหลังจากที่นำระบบย่อยที่ผ่านการทดสอบมารวมกันแล้ว

Acception-test: เป็นการทดสอบความถูกต้องและการยอมรับครั้งสุดท้ายจาก user ก่อนการส่งมอบ เช่น การทดสอบส่วน user interface

2.4.1.4. Evolution คือพัฒนาการของ Software ที่มีการปรับเปลี่ยนไปตามความต้องการของผู้ใช้ หรือความเหมาะสมให้มีประสิทธิภาพมากขึ้น เป็นขั้นตอนการตรวจสอบระบบที่พัฒนาอยู่ในปัจจุบันมาวิเคราะห์กับความต้องการที่ระบบได้ผ่านการวิเคราะห์และลงความเห็นว่าเป็นความต้องการจากผู้ใช้ที่ถูกต้อง เทียบกับความต้องการจากผู้ใช้ที่เกิดการเปลี่ยนแปลงในระหว่างการพัฒนา เพื่อให้ทราบว่าระบบมีข้อบกพร่องเมื่อเทียบกับความต้องการเดิมหรือไม่ หรือระบบควรมีการปรับปรุงเปลี่ยนแปลงหรือมีส่วนไหนบ้างที่จำเป็นต้องเพิ่มเติม ซึ่งจะได้ความต้องการใหม่และจะเป็นกระบวนการกระทำซ้ำจนกว่าจะได้ความต้องการที่ถูกต้องสมบูรณ์

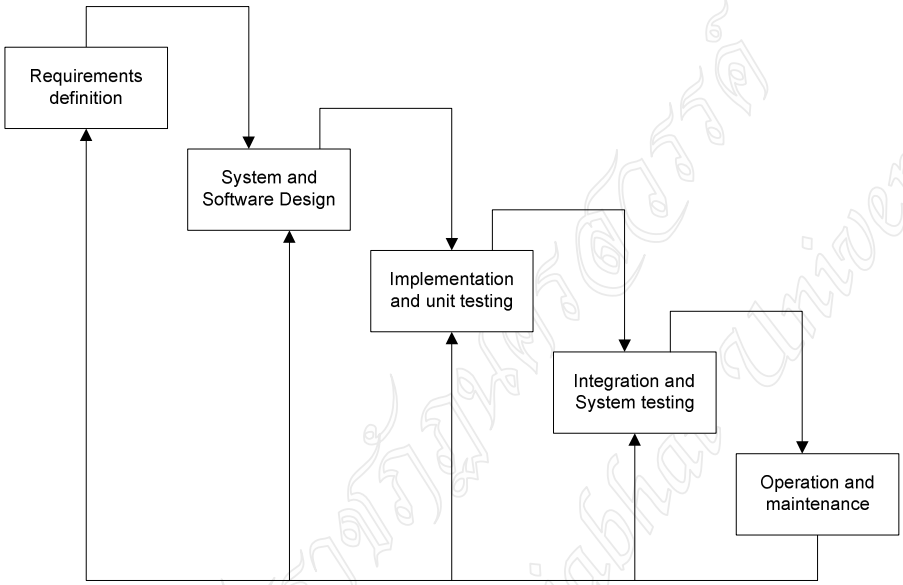
2.4.2 กระบวนการทางซอฟต์แวร์ทั่วไป (Generic Software Process)

ส่วนใหญ่ที่นิยมใช้กันอย่างแพร่หลาย มี อยู่ 3 กระบวนที่สำคัญ

2.4.2.1 Waterfall Model แบบจำลองน้ำตก บางครั้งถูกเรียกว่า วงจรแบบฉบับ (Classic Life Cycle) ซึ่งหมายถึง แบบระเบียบวิธีเรียงลำดับเป็นระบบในการพัฒนาซอฟต์แวร์ เริ่มด้วยการกำหนดความต้องการของลูกค้า และก้าวหน้าไปสู่การวางแผน การสร้างแบบจำลอง การสร้างซอฟต์แวร์ ตามด้วยการให้การช่วยเหลือในการใช้งาน ซึ่งแบบจำลองน้ำตก มี 5 ขั้นตอนดังนี้

- Requirement Definition: เป็นการนิยามหรือกำหนดความต้องการของระบบและคุณสมบัติ
- System and Software Design: การออกแบบซอฟต์แวร์และการออกแบบระบบ
- Implementation and Unit testing: การพัฒนาตามที่ได้ออกแบบและการทดสอบระบบ

- Integration and System testing: การนำระบบย่อยมารวมกันและทดสอบระบบทั้งหมด
- Operation and Maintenance: การติดตั้งใช้งานจริงและขั้นตอนการบำรุงรักษา โดยที่สามารถย้อนกลับไปทำขั้นตอนก่อนหน้าเมื่อพบปัญหา



รูปที่ 2.3 The software life cycle กระบวนการออกแบบและพัฒนาระบบแบบ Waterfall

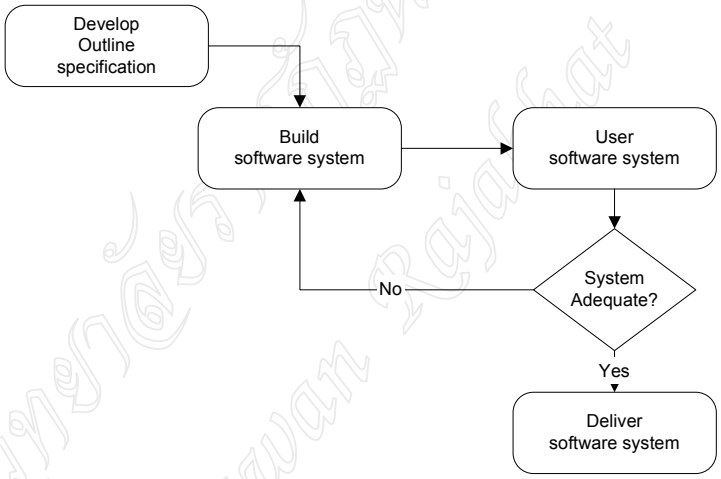
วิธีการนี้ได้รับการพัฒนาขึ้นมาเป็นโมเดลแรกเริ่มตั้งแต่ที่มีการพัฒนาซอฟต์แวร์ โดยเทคนิคนี้จะทำการแบ่งการดำเนินงานของระบบทั้งหมดเป็นขั้นตอนย่อยๆ ที่จำเป็น เช่น การกำหนดความต้องการ การออกแบบระบบ การพัฒนาและการทดสอบระบบ ซึ่งในแต่ละขั้นตอนจะสามารถย้อนกลับไปทำยังขั้นตอนก่อนหน้าได้เมื่อพบปัญหา จึงมีลักษณะการทำงานเป็นขั้นๆ เหมือนกับน้ำตก จึงเรียกระบวนการพัฒนาแบบนี้ว่า waterfall ซึ่งทำให้เกิดวงจรชีวิตของซอฟต์แวร์

2.4.2.2 การพัฒนาแบบ Evolutionary การพัฒนา Softwareจัดเป็นวิวัฒนาการ โดยมีการปรับเปลี่ยน Requirement Specification (Spec.) ไปเรื่อยๆ และพัฒนาปรับเปลี่ยน Specification เป็นลักษณะของ Version

ขั้นแรกเป็นการกำหนดขอบเขตก่อน จากนั้นค่อยกำหนด Spec. ขึ้นมา และพร้อมที่จะพัฒนาได้ในระยะเวลาอันสั้นและทำการตรวจสอบว่า Version แรกถูกต้องและมีอะไรปรับเปลี่ยนหรือไม่ หลังจากได้ version 1 แล้ว เราก็ทำการเพิ่ม Specification ต่างๆ ลงไปหรือการทำงานในมุมต่างๆที่เราเคยตัดทิ้งไปในตอนแรก เราก็จะรวมเข้าไปและทำการพัฒนาต่อไปอีกได้เป็น Version ถัดไป และทำเป็นกระบวนการซ้ำจนกว่าจะได้ Version ที่สมบูรณ์แล้ว

เทคนิคการออกแบบและพัฒนาซอฟต์แวร์นี้ ตั้งอยู่บนแนวความคิดที่จะพัฒนาระบบซึ่งจะทำงานกับระบบที่ทำงานเร็วที่สุดเท่าที่จะเป็นไปได้ให้เสร็จก่อน หลังจากนั้นจึงจะทดลองใช้และปรับปรุงแก้ไขจนกว่าระบบจะทำงานได้ตรงตามความต้องการ ตัวอย่างที่เห็นได้ชัดคือ ในกระบวนการสร้างระบบ “ปัญญาประดิษฐ์” (Artificial Intelligence: AI) ซึ่งเป็นระบบที่เราไม่สามารถกำหนดรายละเอียดเบื้องต้นทุกอย่างได้ ในการออกแบบระบบงานโดยเทคนิคนี้ผู้ออกแบบระบบจะคำนึงถึงความเหมาะสมของการทำงาน (adequacy) มากกว่าความสมบูรณ์ถูกต้อง (correctness)

ดังนั้นมีผู้โต้แย้งว่าทุกระบบงานก็จะเริ่มต้นจากโครงร่างคร่าวๆ ทั้งนั้น แต่อย่างไรก็ตามการออกแบบ exploratory programming นี้เหมาะอย่างยิ่งกับการออกแบบระบบที่เราไม่รู้ว่ารายละเอียดควรเป็นอย่างไร



รูปที่ 2.4 กระบวนการออกแบบและพัฒนาระบบแบบ Exploratory programming

การออกแบบพัฒนาระบบซอฟต์แวร์โดยใช้เทคนิคนี้ อาจจะเหมาะสมสำหรับระบบงานบางอย่างโดยเฉพาะ แม้ว่าอาจใช้ไม่ได้ในการพัฒนาระบบงานทั่วๆ ไป แต่อย่างไรก็ตามเทคนิคนี้ก็อาจนำมาประยุกต์ใช้ได้ดีในการบำรุงรักษาระบบงานทั่วไปที่ยังไม่มีความชัดเจนว่าจะใช้วิธีการใดในการปรับแต่งแก้ไขปัญหาที่เกิดขึ้น โดยสรุปแล้วเทคนิค Exploratory programming เหมาะที่จะถูกเลือกใช้ใน 3 กรณีหลักดังต่อไปนี้

1. โครงสร้างของระบบงานที่กำลังพัฒนาเป็นไปในลักษณะที่ต้องมีการนำไปทดลองใช้อย่างสม่ำเสมอ การปรับแต่งแก้ไขเพื่อความเหมาะสมจึงถือได้ว่าเป็นการประเมินความก้าวหน้าของโปรแกรมไตามลำดับ

2. ระบบงานที่แม้จะมีโครงสร้างที่ไม่ชัดเจน แต่ผู้พัฒนาสร้างระบบและผู้แก้ไขระบบภายหลัง

เป็นบุคคลเดียวกัน แต่อย่างไรก็ตามในกรณีที่เป็นระบบงานใหญ่ และต้องใช้ภายในหน่วยงานเป็นเวลานาน เทคนิคนี้จะไม่เหมาะสมเพราะคนสร้างระบบและคนปรับแต่งแก้ไขระบบในภายหลังมักจะไม่ใช่คนเดียวกัน ซึ่งอาจมีผลทำให้โครงสร้างและรูปแบบของระบบบิดเบือนไปจากจุดเดิมจนเสียรูปได้

3. แม้ว่ายังไม่มีงานวิจัยยืนยันลักษณะของทีมงานที่เหมาะสมสำหรับการออกแบบโดยใช้เทคนิคนี้ แต่มีผู้แนะนำให้กว้างๆ ว่าทีมงานที่จะใช้วิธีการออกแบบชนิดนี้ได้ดีน่าจะเป็นทีมงานขนาดเล็กที่ประกอบด้วยผู้มีทักษะเชี่ยวชาญ และเป็นผู้ที่มีแรงจูงใจในการทำงานสูง

ข้อดี: เกิดจากการที่เราค่อยๆ ทำทีละ Version ทำให้เกิดการขัดเกลาความต้องการได้หลายรอบ ทำให้ความต้องการ (Requirement) มีความผิดพลาดน้อย

ข้อเสีย: บอกความคืบหน้าของงานได้ลำบาก คือ ยากที่จะบอกว่างานดำเนินการไปถึงไหน หรือ การ Check Milestone ได้ค่อนข้างลำบาก

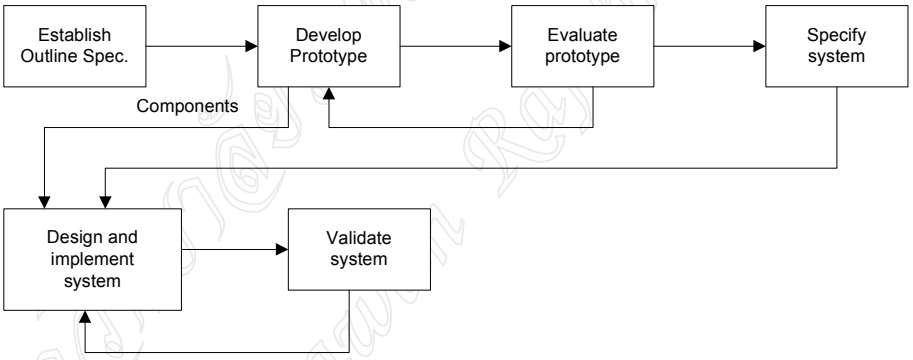
ข้อแตกต่างที่สำคัญระหว่างการออกแบบ Exploratory programming และแบบ Waterfall การออกแบบ Exploratory programming จะเน้นรายละเอียดของระบบตรงขั้นตอนการตรวจสอบและยืนยันความถูกต้อง (V&V) ทั้งนี้เพราะ V แรก (Verification) เป็นการตรวจสอบว่าระบบที่พัฒนาขึ้นมาตรงตามรายละเอียดที่ได้ออกแบบไว้หรือไม่อย่างไร ดังนั้นเมื่อเราไม่สามารถกำหนดรายละเอียดเบื้องต้นได้ ขั้นตอนการตรวจสอบนี้ก็เป็นไปไม่ได้ ส่วนกระบวนการตรวจสอบความถูกต้องของ V ที่สอง (Validation) จึงเป็นการตรวจสอบความถูกต้องเหมาะสมของการทำงานของโปรแกรม มากกว่าเป็นการตรวจสอบความเที่ยงตรงตามโครงสร้างที่ได้ออกแบบไว้

ความถูกต้องเหมาะสมเป็นสิ่งที่วัดได้ยาก และค่อนข้างเป็นการติดสินใจตามผู้ตรวจสอบ (Subjective) เราไม่สามารถรับรองได้ว่าพฤติกรรมของมนุษย์เป็นความถูกต้อง แต่สามารถบอกได้ว่า เราพอใจกับพฤติกรรมที่ทำงานสอดคล้องกับงานที่ต้องการจะให้ทำ

การสร้างแบบจำลอง (Prototyping) เป็นเทคนิคการออกแบบลักษณะ exploratory programming และลักษณะ prototyping มีส่วนที่เหมือนกันคือ เริ่มต้นจากการพัฒนาและนำเสนอระบบงานให้ผู้ใช้ทดลองใช้ แม้ว่าระบบยังไม่สมบูรณ์และยังต้องแก้ไขและปรับเสริมแต่งเพิ่มเติมภายหลัง แต่วิธีการทั้งสองก็มีข้อแตกต่างกันที่สำคัญ คือ exploratory programming เริ่มต้นจากความเข้าใจในรายละเอียดของระบบที่ยังไม่ชัดเจน หลังจากทดลองใช้แล้วจึงมีการเสริมแต่งระบบงานจนเป็นระบบที่ใช้งานได้ดี (executable system) แม้ว่าบางระบบอาจไม่สามารถระบุรายละเอียดได้เลยก็ตาม ส่วนวิธีการแบบ prototyping เริ่มจากโครงร่างของระบบอย่างคร่าวๆ (outline requirements) เช่นเดียวกัน แต่การทดลองสร้างระบบขึ้นมาใช้งานมีจุดมุ่งหมายที่จะค้นหารายละเอียดของระบบ (system specification) ที่พึงประสงค์อาจเรียกได้ว่าเป็นต้นแบบ (prototype) เมื่อได้นำระบบต้นแบบแรกมาทดลองประเมิน และ

ปรับแต่เพิ่มเติมเป็นระบบต้นแบบในลำดับต่อๆ มาตามลำดับจนเห็นว่าใช้การได้แล้ว (executable) อาจทิ้งระบบต้นแบบสุดท้ายนั้น แล้วสร้างระบบงานจริงใหม่อีกครั้งจากต้นแบบที่ได้รับผ่านมา เพื่อให้ได้ระบบงานที่มีคุณภาพตรงตามความต้องการที่แท้จริงของผู้ใช้ระบบ

กระบวนการพัฒนาซอฟต์แวร์เป็นการขยายขั้นตอนการดำเนินการแบบต้นแบบ (Prototyping) โดยมีจุดมุ่งหมายที่จะลดค่าใช้จ่ายในการพัฒนาระบบงาน รูปแบบของระบบต้นแบบตัวอย่างจะมีลักษณะแบบใช้แล้วทิ้ง (throw-away) หลังจากได้ทดลองใช้ศึกษาคุณลักษณะของระบบที่ต้องการแล้วแสดงขั้นตอนการดำเนินงานตามลำดับคือ การพัฒนาระบบตัวอย่าง จะถูกพัฒนามาจากโครงร่างของระบบอย่างง่าย (outline specification) เมื่อระบบตัวอย่างดังกล่าวถูกนำมาศึกษา ทดลองใช้และประเมินผล จนผู้ใช้รู้สึกพอใจในขั้นตอนและวิธีการทำงานแล้ว กระบวนการพัฒนาซอฟต์แวร์ตามปกติก็จะเริ่มต้น โดยการนำเอารายละเอียดของระบบ (system specification) และโครงสร้างหลักของระบบต้นแบบตัวอย่าง (prototype components) มาประกอบการดำเนินการสร้างระบบงานที่ต้องการขึ้นมา และระบบงานใหม่ดังกล่าวนี้จะถูกทดสอบความถูกต้องสมบูรณ์ (validate) จนกว่าจะเป็นที่ยอมรับได้



รูปที่ 2.5 ขั้นตอนดำเนินการของกระบวนการต้นแบบ

ในบางกรณีมีการนำเอาระบบงานต้นแบบ มาจัดเข้าเป็นส่วนหนึ่งของระบบที่สร้างขึ้นมาโดยไม่ต้องดำเนินการจัดสร้างใหม่ ซึ่งเป็นการลดขั้นตอนการพัฒนาสร้างระบบใหม่จากระบบต้นแบบ นั่นคือไม่ต้องมีการสร้างส่วนของระบบงานที่เหมือนกับระบบตัวอย่างต้นแบบที่ได้ทดลองแล้วซ้ำใหม่อีกครั้ง แต่การดำเนินการดังกล่าวนี้อาจมีปัญหาและข้อควรระวังอยู่บ้าง คือ

1. ลักษณะสำคัญบางอย่างของระบบ อาจถูกละเว้นไปในขณะทำเป็นระบบงานต้นแบบเพื่อทำให้การทดลองสะดวกและทำงานได้
2. อาจมีปัญหาทางกฎหมายในเรื่องของการนำเอาระบบต้นแบบไปใช้งานในที่ทำงานจริง และยังคงเป็นปัญหาในเรื่องของลิขสิทธิ์
3. ผู้ใช้ระบบอาจใช้ระบบต้นแบบไม่เหมือนกับตอนที่ระบบงานถูกพัฒนาแล้วและใช้งานจริง

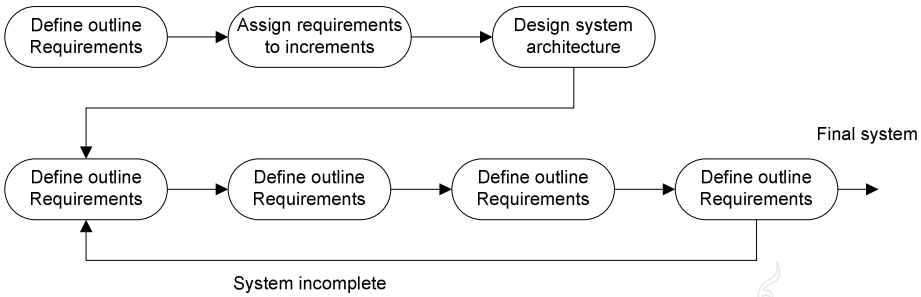
ตัวอย่างเช่น ในขณะที่เป็นระบบงานต้นแบบช่วงเวลาในการทำงานอาจช้าและผู้ใช้ระบบปรับตัวจนเคยชินกับระบบต้นแบบแล้ว เมื่อพัฒนาเป็นระบบงานจริงเราต้องการทำงานที่เร็วขึ้น ผู้ใช้ระบบอาจต้องเผชิญหน้ากับความไม่คุ้นเคยและทำให้ระบบไม่ประสบผลสำเร็จเท่าที่ควร

ด้วยสาเหตุดังกล่าวข้างต้น ในการพัฒนาระบบงานใหญ่ๆ จึงควรมีการสร้างและพัฒนาระบบงานจริงขึ้นมาใหม่จากระบบงานต้นแบบที่ได้ทดลองใช้ โดยไม่ควรนำเอาระบบต้นแบบมาพัฒนาใช้งานเลย (เพื่อลดค่าใช้จ่าย) ทั้งนี้ด้วยเหตุผลดังนี้

1. ลักษณะการทำงานบางอย่าง เช่น พฤติกรรมของระบบและผู้ใช้ระบบ ความมั่นคงปลอดภัยของระบบ ความน่าเชื่อถือของข้อมูลในระบบอาจถูกมองข้ามไปอย่างจงใจในระบบงานต้นแบบเมื่อเวลาทดลองใช้ แต่ในระบบงานจริงจะละเว้นลักษณะของระบบดังกล่าวไม่ได้ และหลายๆ กรณีเราไม่สามารถเพิ่มลักษณะดังกล่าวเข้าในระบบต้นแบบ
2. ในขณะพัฒนาระบบงานต้นแบบอาจมีการเปลี่ยนแปลงรูปแบบอยู่ตลอดเวลาเพื่อให้ตรงตามความต้องการของผู้ใช้ระบบ การเปลี่ยนแปลงดังกล่าวอาจเกิดขึ้นในสภาพที่ไร้การควบคุม มีการเปลี่ยนรูปแบบไปเรื่อยๆ จนกว่าจะพอใจ และเมื่อนำเอาระบบต้นแบบที่พอใจแล้วมาใช้งานจริง ก็อาจจะประสบปัญหาการบำรุงดูแลรักษาระบบในภายหลังได้
3. การเปลี่ยนแปลงระบบต้นแบบบ่อยครั้ง อาจเป็นการลดโครงสร้างที่จำเป็นต้องมีในระบบงานจริง เพื่อทำให้สะดวกและง่ายมักจะทดลองคุณลักษณะนั้นๆ ของระบบการเปลี่ยนแปลงมากเกินไปอาจทำให้เราไม่สามารถบำรุงรักษาระบบต้นแบบนั้นในภายหลัง

การใช้ prototype ทำให้เกิดประโยชน์อย่างยิ่งในการนำไปทำความเข้าใจเกี่ยวกับทิศทางที่จะออกแบบระบบรวมและ interface ของระบบ โดยมีการปรับเปลี่ยนไปตามความต้องการให้มีประสิทธิภาพ (Nauman and Jenkins, 1982)

2.4.2.3 การพัฒนาแบบเพิ่มขึ้น (Incremental Development) การพัฒนาแบบเพื่อขึ้น (Incremental Development) จะมีลักษณะที่สอดคล้องกับ Model ต่างๆ ที่กล่าวไว้เป็นส่วนใหญ่ โดยแนวคิดของการพัฒนาแบบเพื่อขึ้น คือ แทนที่เราจะทำ Software ให้จบทีเดียว เราก็จะทำเป็นทีละส่วนย่อยๆ แล้วนำไปทำการทดสอบหรือไม่ก็ส่งงานไปเหมือนกับเป็น Milestone ย่อยๆ ไป



รูปที่ 2.6 การพัฒนาแบบ Incremental

จากรูปที่ 2.6 ในขั้นตอนแรกจะเป็นการกำหนด Requirement คร่าวๆ แล้วขั้นต่อมาจะพิจารณาหาส่วนที่จะเพิ่ม requirement เข้าไป หลังจากนั้นก็จะเป็นขั้นตอนการออกแบบให้เห็นภาพรวมของระบบแล้วจึงมาออกแบบงานชิ้นย่อยชิ้นแรกโดยละเอียดและตรวจสอบ แล้วก็วนกลับมาทำ incremental ที่ 2 แล้วตรวจสอบ แล้วนำมารวมกับ increment ที่ 1 แล้วตรวจสอบภาพรวมว่าทำงานได้เป็นอย่างไร แล้วจึงวนกลับไปทำ increment ที่ 3 และ increment ที่เหลือซ้ำจนจบทุก increment ที่กำหนดไว้จนได้ออกมาเป็น Product สุดท้าย

ข้อดี:

1. increment แรกๆ จะโดนตรวจสอบหลายรอบ
2. ถ้า requirement เปลี่ยนแปลงก็สามารถนำรวมเข้าไปได้เรื่อยๆ