บทที่ 3

ทฤษฎีที่เกี่ยวข้องในวิศวกรรมซอฟต์แวร์

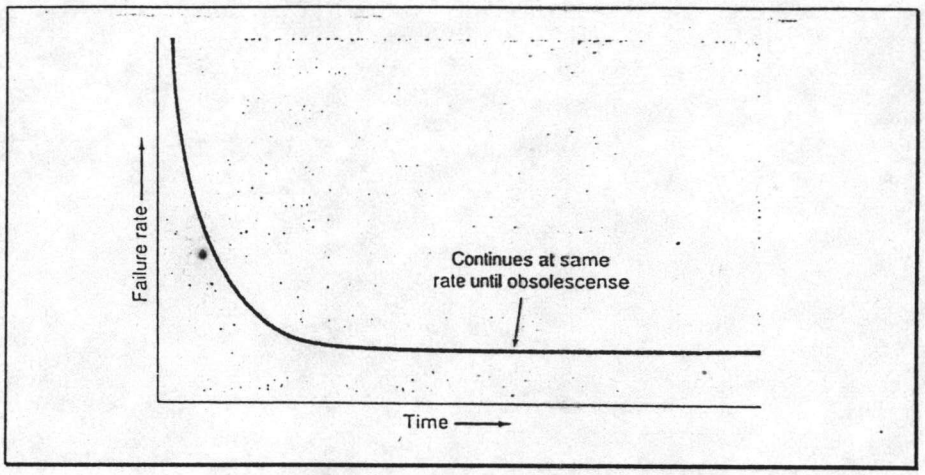
ในยุคก่อนหน้านี้ การบริหารงานพัฒนาระบบคอมพิวเตอร์มักจะพิจารณาทางด้านของฮาร์ดแวร์เป็นหลักเนื่องจากเป็นส่วนที่มีราคาสูงมาก ได้มีการพัฒนาเทคนิคมาตรฐานต่างๆขึ้นมาใช้ในการวัดประสิทธิภาพและการปรับปรุง และทอดทิ้งส่วนของซอฟต์แวร์โดยพิจารณาว่าเป็นงานศิลปะ มีการพัฒนาเทคนิคต่างๆกันน้อย โปรแกรมเมอร์เรียนรู้งานโดยใช้วิธีการลองผิดลองถูก แต่ในปัจจุบันการกระจายของค่าใช้จ่ายในระบบคอมพิวเตอร์เปลี่ยนแปลงไป โดยซอฟต์แวร์กลับมีค่าใช้จ่ายที่สูงกว่า จึงมีการพัฒนาเทคนิคทางด้านซอฟต์แวร์ต่างๆขึ้นมาเกิดเป็นวิชาทางด้านวิศวกรรมซอฟต์แวร์

ลักษณะของซอฟต์แวร์เป็นสิ่งที่เป็นตรรกะ มีความแตกต่างกับฮาร์ดแวร์ซึ่งเป็นกายภาพดังนี้

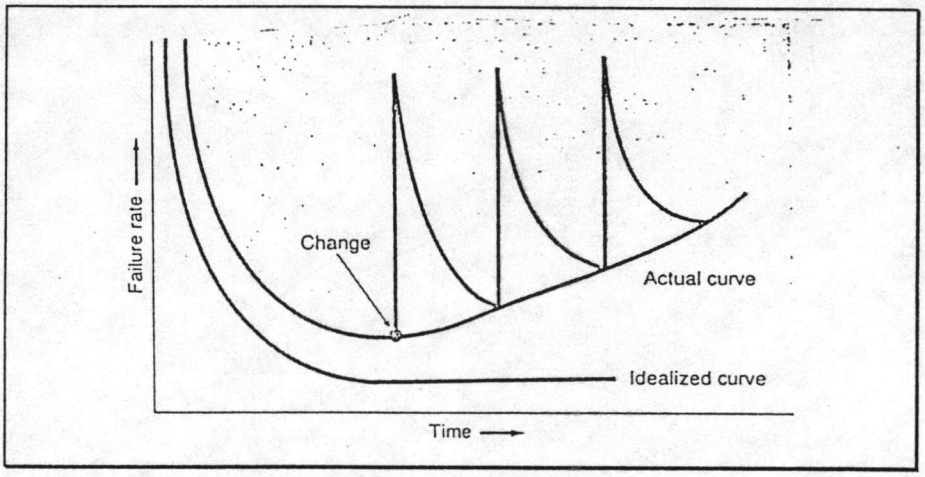
1. ซอฟต์แวร์เป็นการพัฒนาและวิศวกรรมไม่ใช่งานการผลิต ทั้งสองอย่างคุณภาพที่สูงเริ่มจากการออกแบบที่ดี แต่ในซอฟต์แวร์จะไม่มีปัญหาของคุณภาพในช่วงของการผลิต มีการใช้คนทำงานในลักษณะที่ต่างกัน

2. ซอฟต์แวร์ไม่มีการเสียแล้วทิ้ง ในฮาร์ดแวร์เมื่ออายุการใช้งานมากขึ้นโอกาสที่จะเสียมีมากขึ้นและเปลี่ยนใหม่ในที่สุด แต่ในซอฟต์แวร์เมื่อมีปัญหาเกิดขึ้นและได้รับการแก้ไขแล้วปัญหานั้นจะไม่เกิดขึ้นอีก ดังกราฟที่แสดงในรูปที่ 3.1 แต่ในความเป็นจริงแล้ว เมื่อใช้งานไประยะหนึ่งจะมีการปรับปรุงหรือพัฒนาเพิ่มความสามารถให้ซอฟต์แวร์ ทำให้เกิดปัญหาใหม่ขึ้นมาและเมื่อได้รับการแก้ไขก็จะกลับเข้าสู่จุดสมดุลของอัตราการผิดพลาด เป็นวัฏจักร ดังแสดงในกราฟรูปที่ 3.2

3. ซอฟต์แวร์ส่วนใหญ่สร้างขึ้นมาเองโดยไม่ได้นำมาจากส่วนสำเร็จรูป ซึ่งต่างกับการออกแบบฮาร์ดแวร์ที่มีการออกแบบโดยใช้อุปกรณ์ที่มีขายตามท้องตลาด ในขณะที่การออกแบบซอฟต์แวร์ไม่มีส่วนประกอบขาย แม้จะมีก็เป็นลักษณะของส่วนสำเร็จ



รูปที่ 3.1 กราฟความผิดพลาดของซอฟต์แวร์ในอุดมคติ

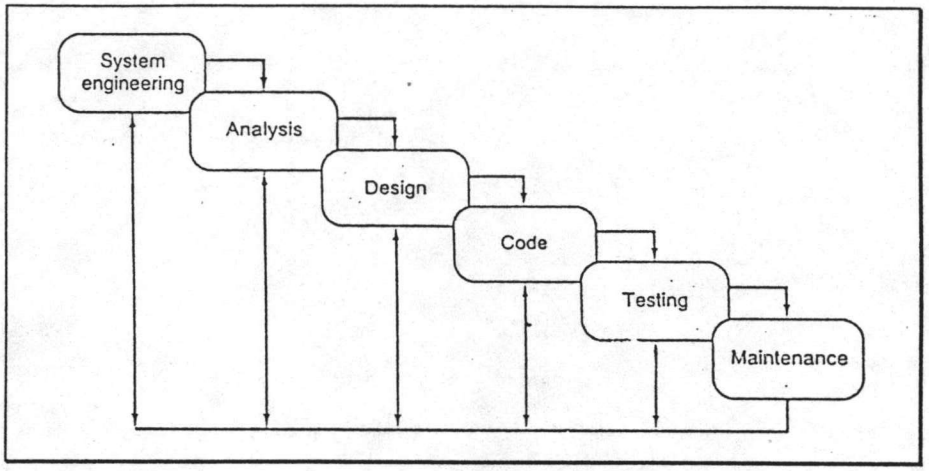


รูปที่ 3.2 กราฟความผิดพลาดของซอฟต์แวร์ในความจริง

เทคนิคในการพัฒนาซอฟต์แวร์ในปัจจุบันยังไม่มีวิธีการที่ดีที่สุด ได้มีการศึกษาวิธีการสำหรับงาน ในแต่ละช่วงของการพัฒนา การสร้างเครื่องมืออัตโนมัติในขั้นตอนต่างๆ การประกันคุณภาพ การประสานงาน การควบคุม และการบริหารงาน ปัจจัยแต่ละด้านของการพัฒนาซอฟต์แวร์นี้รวมเรียกว่าวิศวกรรมซอฟต์แวร์

<u>วัฏจักรของซอฟต์แวร์</u>

การศึกษาวัฏจักรของซอฟต์แวร์เพื่อให้การพัฒนาซอฟต์แวร์เป็นไปอย่างมีระบบและขั้นตอน แบบจำลองที่นิยมใช้กันคือ แบบจำลองน้ำตก (Waterfall Model) ดังแสดงในรูปที่ 3.3 แบ่งเป็นขั้นตอนต่างๆดังนี้



รูปที่ 3.3 วัฏจักรของซอฟต์แวร์

1. การศึกษาและวิเคราะห์ระบบ (System Engineering and Analysis)

เนื่องจากซอฟต์แวร์มักจะเป็นส่วนประกอบส่วนหนึ่งของระบบงานทั้งหมด โดยเริ่มต้นจากการศึกษาความต้องการของระบบโดยรวมทั้งหมด และจัดสรรความต้องการบางส่วนนี้มาพัฒนาเป็นซอฟต์แวร์ การพิจารณาระบบมีส่วนสำคัญเมื่อซอฟต์แวร์ต้องมีการติดต่อกับส่วนอื่นๆ เช่นฮาร์ดแวร์ คน และฐานข้อมูล การศึกษาและวิเคราะห์ระบบประกอบด้วยการรวบรวมข้อมูล และการวิเคราะห์การไหลของข้อมูล

2. การวิเคราะห์ความต้องการของซอฟต์แวร์ (Software Requirement Analysis)

การรวบรวมข้อมูลความต้องการของระบบจะลดขอบเขตให้มาอยู่ในระดับของซอฟต์แวร์ เพื่อที่จะเข้าใจลักษณะของโปรแกรมที่จะสร้าง นักวิเคราะห์จำเป็นที่ต้องเข้าใจส่วนหลักของสารสนเทศ หน้าที่ที่ต้องการ ประสิทธิภาพ และตัวประสานกับผู้ใช้ มีการบันทึกความต้องการของระบบและซอฟต์แวร์และนำมาตรวจทานกับลูกค้าต่อไป

3. การออกแบบ (Design)

การออกแบบซอฟต์แวร์มีขั้นตอนต่างๆหลายขั้น ซึ่งมีขั้นที่สำคัญคือ โครงสร้างข้อมูล สถาปัตยกรรมของซอฟต์แวร์ รายละเอียดของกระบวนการ และตัวประสาน ขั้นตอนการออกแบบจะเปลี่ยนความต้องการให้เป็นสิ่งนำเสนอได้ เพื่อให้สามารถประเมินผลได้ก่อนที่จะเริ่มการเขียนโปรแกรม มีการบันทึกเป็นเอกสารและใช้เป็นส่วนหนึ่งของโครงแบบของระบบ

4. การเขียนโปรแกรม (Coding)

เป็นการนำสิ่งที่ออกแบบเสร็จมาเปลี่ยนเป็นรูปแบบที่เครื่องสามารถเข้าใจได้ ถ้ามีการออกแบบที่ดี มีรายละเอียดครบถ้วน ก็ทำให้การเขียนเป็นไปได้รวดเร็ว

5. การทดสอบ (Testing)

เมื่อเขียนโปรแกรมเสร็จก็จะมีการทดสอบ ขั้นตอนการทดสอบจะมุ่งเน้นในส่วนของตรรกะภายในของระบบเพื่อประกันว่าทุกคำสั่งของโปรแกรมได้รับการทดสอบ และการทดสอบหน้าที่ภายนอกเพื่อดูความผิดพลาดที่ไม่ครอบคลุม และมั่นใจว่าข้อมูลที่รับเข้าได้ผลตรงตามที่ต้องการ

6. การบำรุงรักษา (Maintenance)

เมื่อส่งมอบซอฟต์แวร์ให้กับลูกค้าแล้ว มักจะมีการเปลี่ยนแปลงเสมอ การเปลี่ยนแปลงอาจเกิดเนื่องมาจากความผิดพลาดจากการเปลี่ยนสิ่งแวดล้อมภายนอก หรือความต้องการที่เพิ่มขึ้นของลูกค้า การบำรุงรักษาทำให้เกิดการกลับไปสู่ขั้นตอนในวัฏจักรก่อนหน้านี้

แบบจำลองนี้เป็นแบบจำลองที่สร้างมานานที่สุดและมีการใช้อย่างกว้างขวาง ปัญหาเมื่อเปรียบเทียบกับการทำงานจริง มีดังนี้

1. ในโครงการจริงไม่เป็นไปตามลำดับตามที่เสนอ มีการทำซ้ำกลับไปมาเสมอ
2. การที่ลูกค้าจะระบุความต้องการให้ชัดเจนทำได้ยาก ซึ่งเป็นข้อกำหนดของวัฏจักร และไม่สามารถรวบรวมความต้องการต่างๆได้ตั้งแต่ตอนเริ่มต้นโครงการ
3. ลูกค้าต้องมีความอดทน เนื่องจากระบบที่ใช้งานได้จะมีให้ใช้จนกว่าถึงขั้นตอนสุดท้ายของโครงการ ทำให้ปัญหาที่สำคัญอาจจะไม่พบจนกว่าจะใช้งาน แล้วเกิดความเสียหายได้

ปัญหาที่กล่าวมาเป็นสิ่งที่เกิดขึ้นจริง แต่วัฏจักรของซอฟต์แวร์ก็เป็นสิ่งที่สำคัญของงานวิศวกรรมซอฟต์แวร์ เพราะใช้เป็นแนวทางสำหรับการทำงาน และยังคงเป็นวัฏจักรที่มีการใช้อย่างกว้างขวาง

โดยทั่วไปลูกค้ามักจะกำหนดเป้าหมายกว้างๆของซอฟต์แวร์ไว้ แต่ไม่กำหนดรายละเอียดของส่วนรับ การประมวลผล และสิ่งออก หรือว่าผู้พัฒนาเองไม่แน่ใจในประสิทธิภาพ ขั้นตอนวิธี ความปรับต่อของระบบปฏิบัติการ หรือลักษณะของการติดต่อระหว่างเครื่องกับคน ในสภาพต่างๆนี้การทำต้นแบบของวิศวกรรมซอฟต์แวร์จะช่วยแก้ไขได้

การทำต้นแบบเป็นขั้นตอนเพื่อช่วยให้นักพัฒนาสร้างแบบจำลองของซอฟต์แวร์ สำหรับที่จะสร้างต่อไป ประกอบด้วย 3 แบบคือ

1. ต้นแบบโดยใช้กระดาษหรือเครื่องพีซี เพื่อแสดงการติดต่อกัน เพื่อให้ผู้ใช้เข้าใจการทำงาน
2. การทำต้นแบบโดยการดึงเอาบางส่วนจากงานจริงมาทำ
3. จากโปรแกรมเดิมที่มีอยู่แล้ว ซึ่งมีความสามารถตามความต้องการหลัก และปรับปรุงเพิ่มขึ้นในความต้องการที่นอกเหนือจากนี้

<u>ซอฟต์แวร์เมตริก</u>

การบริหารโครงการซอฟต์แวร์จัดเป็นชั้นแรกสุดของกรรมวิธีทางวิศวกรรมซอฟต์แวร์ การพัฒนาโครงการซอฟต์แวร์สำเร็จประกอบด้วยปัจจัยต่างดังนี้

1. การเริ่มต้นโครงการซอฟต์แวร์ ก่อนวางแผนการพัฒนาซอฟต์แวร์ ควรมีการกำหนดจุดประสงค์และขอบเขตให้ชัดเจน พิจารณาทางเลือกของปัญหาต่างๆ กำหนดข้อจำกัดทางด้านเทคนิค และการจัดการ

2. การวัดและเมตริก ในการแก้ปัญหาทางเทคนิค การวัดผลและเมตริกช่วยให้เข้าใจขั้นตอนทางเทคนิคที่ใช้ในการพัฒนาและผลิตภัณฑ์เอง การวัดผลการพัฒนาช่วยในการปรับปรุงการทำงาน และการวัดผลผลิตภัณฑ์ช่วยให้มีคุณภาพสูงขึ้น

3. เมื่อเข้าสู่การวางแผนการพัฒนาซอฟต์แวร์ จำเป็นต้องมีการประมาณต่างๆเช่น จำนวนคน ระยะเวลา และค่าใช้จ่ายที่ใช้ ในยุคแรกการประมาณต่างๆใช้ข้อมูลจากประสบการณ์เป็นแนวทาง ซึ่งถ้าเป็นโครงการที่ไม่เคยทำมาก่อนจะทำได้ยาก จึงมีการพัฒนาเทคนิคที่ใช้ในการประมาณ โดยใช้ข้อมูลพื้นฐานที่เหมือนกันคือ

- ขอบเขตของโครงการที่ระบุชัดเจน
- ซอฟต์แวร์เมตริกโดยการใช้ข้อมูลในอดีต และใช้เป็นข้อมูลพื้นฐานในการประมาณ
- แบ่งโครงการออกเป็นส่วนย่อยและประมาณเป็นส่วน

4. การวิเคราะห์ความเสี่ยง เมื่อมีการทำโครงการพัฒนาซอฟต์แวร์ ทำให้มีความเสี่ยงต่างๆเกิดขึ้น จึงมีการวิเคราะห์ความเสี่ยง

5. การจัดลำดับงาน การจัดลำดับของโครงการซอฟต์แวร์มีลักษณะเช่นเดียวกับโครงการวิศวกรรมต่างๆ คือ การกำหนดงานต่างๆในโครงการ ความเกี่ยวข้องกันของแต่ละงาน ระยะเวลาที่ใช้ และการจัดสรรคนกับทรัพยากร

6. การติดตามและควบคุม เมื่อเริ่มต้นโครงการพัฒนาซอฟต์แวร์ ก็ต้องมีการติดตามและควบคุมความก้าวหน้าของงาน ผู้จัดการโครงการติดตามแต่ละงานที่จัดลำดับไว้ ถ้าหากล่าช้าก็ต้องมีการเปลี่ยนแปลงลำดับการทำงาน การจัดการทรัพยากรใหม่ เพื่อให้โครงการดำเนินไปตามแผนที่วางไว้

การวัดเป็นสิ่งพื้นฐานในงานวิศวกรรม การวัดผลในซอฟต์แวร์มีประโยชน์หลายประการคือ

1. ใช้แสดงคุณภาพของผลิตภัณฑ์
2. ใช้ประเมินผลิตผลของคนที่พัฒนา
3. ใช้ประเมินประโยชน์ที่ได้รับจากการใช้เทคนิคทางวิศวกรรมและเครื่องมือใหม่ๆ
4. ใช้เป็นฐานในการประมาณ
5. ใช้ในการปรับปรุงเครื่องมือใหม่และการฝึกอบรม

การวัดผลแบ่งออกเป็น 2 แบบคือ

1. การวัดผลทางตรง

   ประกอบด้วย การวัดจำนวนบรรทัด(Lines of Code - LOC) ความเร็วในการประมวลผล ขนาดหน่วยความจำ และจำนวนรายงานความผิดพลาดในช่วงเวลา

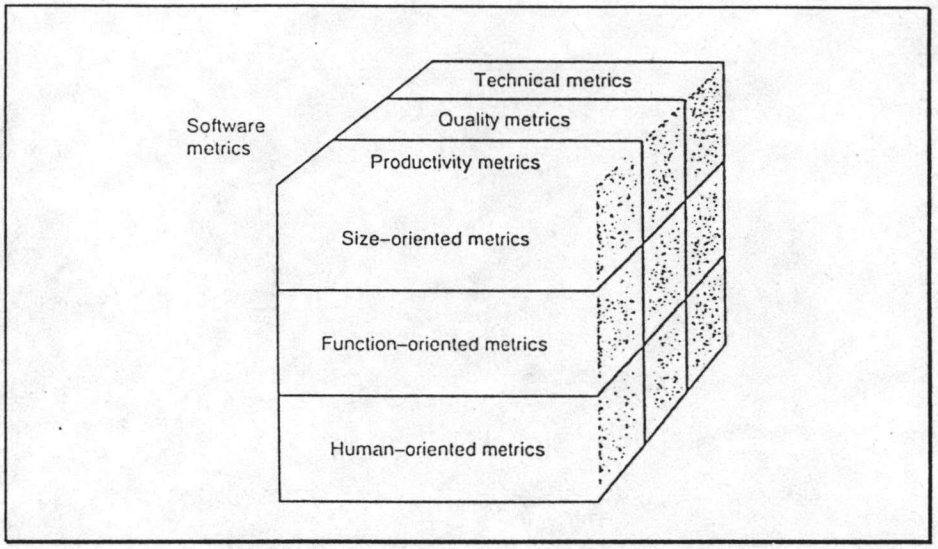
2. การวัดผลทางอ้อม

   ประกอบด้วย ความเป็นฟังก์ชัน(Functionallity) ความซับซ้อน(Complexity) ประสิทธิภาพ ความเชื่อมั่น และการบำรุงรักษา

จะเห็นว่าการวัดผลทางตรงต่างๆสามารถวัดได้ง่ายกว่าการวัดผลทางอ้อม

การแบ่งส่วนประกอบหลักของซอฟต์แวร์เมตริกออกเป็นส่วนดังแสดงในรูปที่ 3.4 ประกอบด้วยส่วนต่างๆคือ

1. ผลผลิต พิจารณาจากผลลัพธ์ของขั้นตอนในวิศวกรรมซอฟต์แวร์
2. คุณภาพ ใช้แสดงผลว่าการทำงานใกล้เคียงกับความต้องการ ทั้งภายในและภายนอก
3. เทคนิค พิจารณาจากลักษณะของซอฟต์แวร์ เช่น ความซับซ้อนทางตรรกะ ดัชนีของการเป็นมอดูล
4. ขนาด เป็นข้อมูลที่ได้จากการวัดผลโดยตรง
5. ฟังก์ชัน เป็นการวัดผลทางอ้อม
6. บุคคล เช่นลักษณะของคนที่พัฒนา การเรียนรู้และความพอใจในเครื่องมือและวิธีการที่ใช้ในการทำงาน



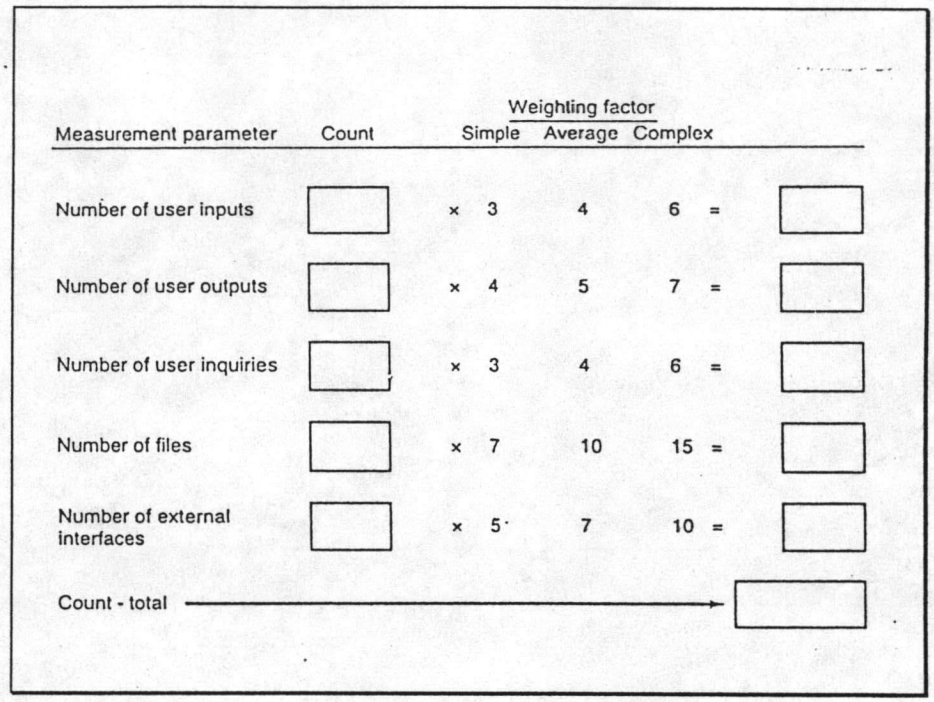
รูปที่ 3.4 ส่วนประกอบของซอฟต์แวร์เมตริก

เมตริกของขนาด การวัดขนาดและกระบวนการที่ใช้ในการพัฒนาโดยตรง เป็นวิธีการวัดที่มีการวิจารณ์กันมาก ทำให้ไม่เป็นที่ยอมรับกันในการวัดผลเพราะขึ้นอยู่กับภาษาที่ใช้และความสามารถในการออกแบบ โดยที่ภาษาระดับที่สูงกว่าและการออกแบบที่ดีย่อมมีจำนวนบรรทัดน้อยกว่า แต่ก็เป็นวิธีที่ง่ายและให้ผลชัดเจน นอกจากนี้ยังนำผลของวิธีนี้ไปใช้วิเคราะห์ในแบบจำลองอื่นๆ

เมตริกของฟังก์ชัน เป็นการวัดผลขนาดและกระบวนการพัฒนาทางอ้อม แทนที่จะนับจากจำนวนบรรทัดของโปรแกรม จะพิจารณาจากการเป็นฟังก์ชันหรืออรรถประโยชน์ วิธีการแรกที่นำเสนอเรียกว่าวิธีฟังก์ชันพอยท์ ซึ่งคำนวณจากจำนวนของข้อมูลที่ใช้ในระบบ และความสัมพันธ์ของข้อมูลกับความซับซ้อนของการประมวลผลข้อมูลนั้น โดยพิจารณาจากปัจจัยต่างๆคือ

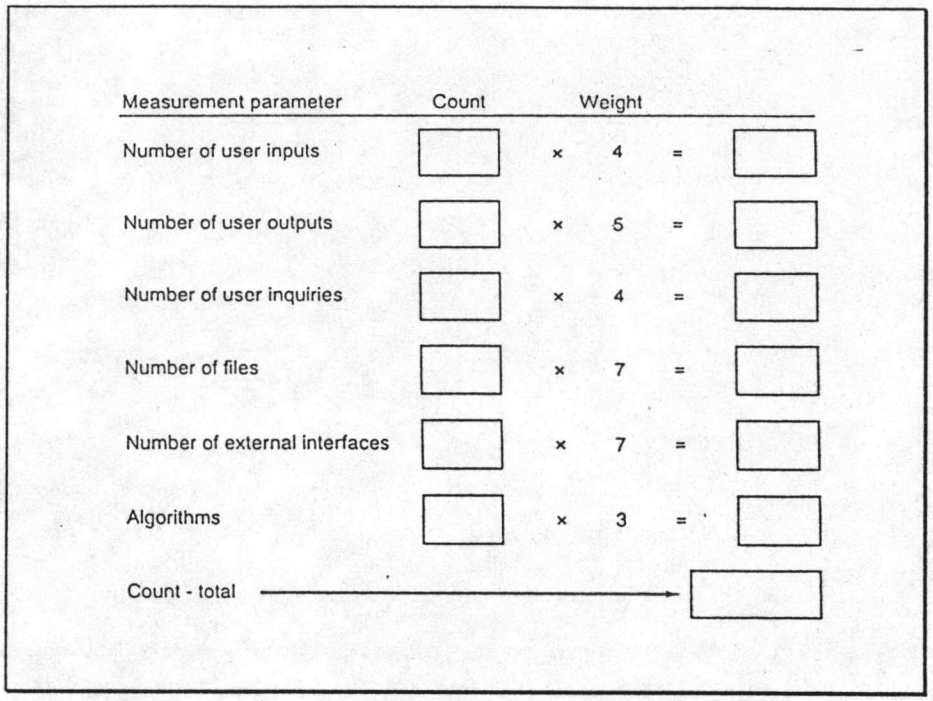
1. จำนวนข้อมูลรับเข้าจากผู้ใช้
2. จำนวนข้อมูลส่งออกให้ผู้ใช้
3. จำนวนคำถามจากผู้ใช้
4. จำนวนแฟ้มข้อมูล
5. จำนวนการประสานกับภายนอก

เมื่อเก็บข้อมูลและนำมาคำนวณดังแสดงในรูปที่ 3.5



รูปที่ 3.5 การคำนวณฟังก์ชันพอยท์

วิธีการฟังก์ชันพอยท์เป็นวิธีที่ออกแบบมาสำหรับการประยุกต์ในระบบทางด้านธุรกิจ และมีการพัฒนาต่อเป็นวิธีฟีเจอร์พอยท์ (Feature Points) โดยเพิ่มปัจจัยขั้นตอนวิธีของโปรแกรมเข้าไป เพื่อให้สามารถนำไปใช้กับการประยุกต์ทางด้านวิทยาศาสตร์และวิศวกรรมได้ ดังแสดงในรูปที่ 3.6



รูปที่ 3.6 การคำนวณฟีเจอร์พอยท์

การวัดผลของ 2 วิธีนี้ใช้ข้อมูลพื้นฐานที่รู้แล้วก่อนการพัฒนา โดยวัดจากความเป็นฟังก์ชันหรืออรรถประโยชน์ของซอฟต์แวร์ และไม่ขึ้นอยู่กับภาษาที่ใช้ จากการวิจัยเพื่อเปรียบเทียบผลการวัดของ 2 วิธีนี้ พบว่าในการประยุกต์ทางด้านสารสนเทศและงานวิศวกรรมให้ค่าเท่ากัน ส่วนในระบบการประมวลผลแบบทันทีที่ซับซ้อนค่าฟีเจอร์พอยท์จะมีค่าสูงกว่าประมาณ 20-35 เปอร์เซนต์

เมตริกของคุณภาพ การวัดคุณภาพของซอฟต์แวร์สามารถวัดได้ในทุกช่วงการทำงานในวัฏจักรซอฟต์แวร์ รวมทั้งหลังจากส่งให้ลูกค้าใช้งานแล้ว การวัดในช่วงก่อนส่งมอบนั้นเป็นการวัดในส่วนของการออกแบบและการทดสอบประกอบด้วย ความซับซ้อนของซอฟต์แวร์ ประสิทธิภาพของสภาพมอดูลาร์ และขนาดของซอฟต์แวร์ทั้งหมด ส่วนช่วงหลังส่งมอบพิจารณาจากจำนวนของข้อบกพร่องที่เกิดขึ้น และการบำรุงรักษาระบบ คุณภาพซอฟต์แวร์หลังส่งมอบจัดเป็นส่วนสำคัญที่ต้องเก็บข้อมูลไว้เพื่อใช้ในการปรับปรุงต่อไป

ความสัมพันธ์ระหว่างจำนวนบรรทัดและค่าฟังก์ชันพอยท์ขึ้นอยู่กับภาษาที่ใช้ในการพัฒนาซอฟต์แวร์ ได้มีการศึกษาเพื่อหาความสัมพันธ์ดังกล่าวดังตารางที่ 3.1 แสดงค่าประมาณของจำนวนบรรทัดต่อ 1 ฟังก์ชันพอยท์ในภาษาต่างๆ

Programming Language	*OC/FP (Average)*
Assembly language	300
COBOL	100
FORTRAN	100
Pascal	90
Ada	70
Object-oriented languages	30
Fourth-generation languages (4GL)	20
Code generators	15

ตารางที่ 3.1 ความสัมพันธ์ระหว่างจำนวนบรรทัดในแต่ละภาษาต่อ 1 ฟังก์ชันพอยท์

นอกจากนี้ยังมีการศึกษาเพื่อหาปัจจัยสำคัญที่มีผลต่อผลิตผลของซอฟต์แวร์ซึ่งประกอบด้วย

1. คน จำนวนและความชำนาญในองค์กร
2. ปัญหา ความซับซ้อนของปัญหา และจำนวนการเปลี่ยนแปลงการออกแบบและความต้องการ
3. กระบวนการ เทคนิคที่ใช้ในการวิเคราะห์และออกแบบ ภาษา เครื่องมือ และการทดสอบที่ใช้ในการพัฒนา
4. ผลิตภัณฑ์ ความเชื่อมั่นและประสิทธิภาพของระบบคอมพิวเตอร์ที่ใช้
5. ทรัพยากร เครื่องมือ ซอฟต์แวร์ และฮาร์ดแวร์ที่จัดหาได้

ผลของปัจจัยต่างๆที่กล่าวมานี้ได้มีการวิจัยเพื่อหาค่าน้ำหนักที่มี แสดงในตารางที่ 3.2

Factor	*Approximate % Variation*
People factors	90
Problem factors	40
Process factors	50
Product factors	140
Resource factors	40

ตารางที่ 3.2 น้ำหนักของปัจจัยที่มีผลกระทบต่อผลผลิตซอฟต์แวร์

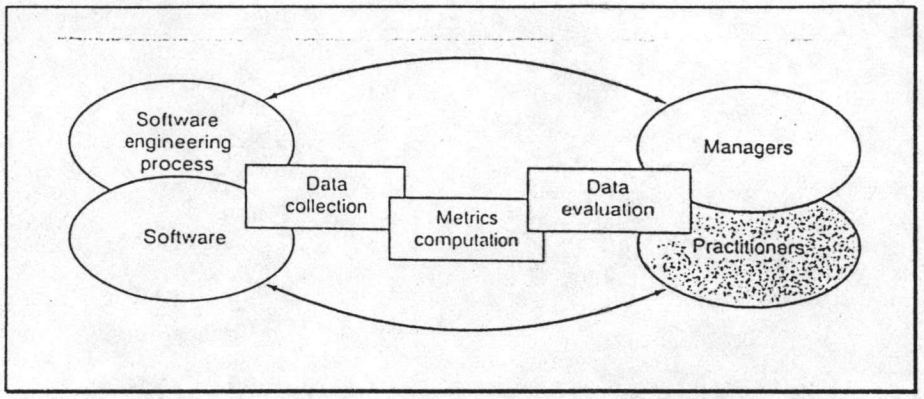
พบว่าวิธีการฟังก์ชันพอยท์และจำนวนบรรทัดให้ความถูกต้องในการพยากรณ์ความพยายามและค่าใช้จ่ายที่ใช้ในการพัฒนาซอฟต์แวร์พอสมควร แต่นักพัฒนาทั่วไปมักจะไม่มีการวัดและไม่เห็นความสำคัญซึ่งการวัดผลให้ประโยชน์ต่างๆ เช่น

1. เป็นข้อมูลในการเปรียบเทียบการปรับปรุงประสิทธิภาพในการทำงาน
2. เป็นส่วนหนึ่งของสิ่งที่มีการปรับปรุงในการบริหารโครงการ
3. การประมาณที่ถูกต้องทำให้การพัฒนาเสร็จตามแผนที่วางไว้ บริหารงานได้ดีขึ้น
4. เป็นข้อมูลทางเทคนิคสำหรับผู้พัฒนา ในส่วนของความถี่ของความต้องการเปลี่ยนแปลง ความผิดพลาดของแต่ละมอดูล จำนวนการทดสอบของแต่ละมอดูล และจำนวนความผิดพลาดที่คาดว่าจะเกิดในการทดสอบ

ลักษณะของเทคนิคการประมาณเพื่อให้ข้อมูลที่ช่วยในการวางแผนและการประมาณค่าใช้จ่ายถูกต้อง มีดังนี้

1. ข้อมูลควรมีความถูกต้อง ไม่ใช้การคาดเดาจากโครงการที่ผ่านมา
2. การเก็บข้อมูลควรมาจากโครงการต่างๆมากที่สุดที่จะทำได้
3. วิธีการวัดมีความแน่นอนในทุกโครงการ
4. การประยุกต์ควรมีลักษณะเช่นเดียวกับข้อมูลที่ใช้ประมาณ

รูปที่ 3.7 แสดงกระบวนการในการกำหนดเกณฑ์ โดยเริ่มจากการเก็บรวบรวมข้อมูลของโครงการที่ผ่านมา นำข้อมูลมาทำการคำนวณค่าเมตริกที่เกี่ยวข้องโดยเฉพาะค่าจำนวนบรรทัดและค่าฟังก์ชันพอยท์เพื่อใช้นำมาเป็นเกณฑ์ในการประมาณโครงการใหม่ที่กำลังจะทำต่อไป

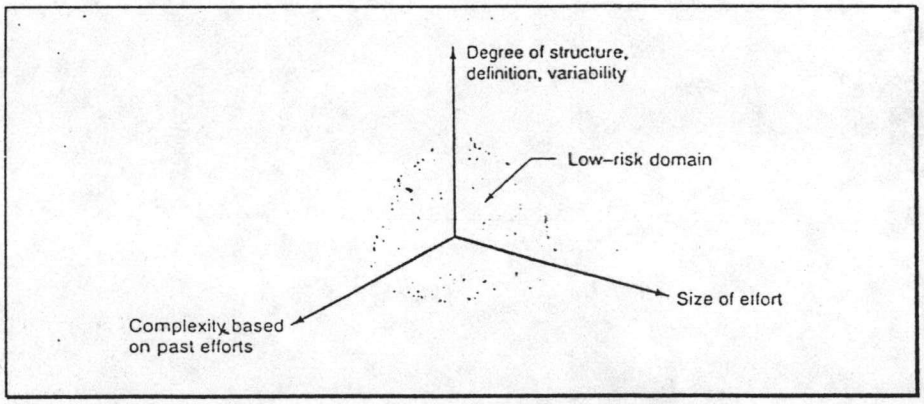


รูปที่ 3.7 กระบวนการในการรวบรวมและวิเคราะห์ข้อมูลของซอฟต์แวร์เมตริก

<u>การประมาณโครงการซอฟต์แวร์</u>

เมื่อกำหนดกิจกรรมที่มีในโครงการเสร็จแล้ว นำข้อมูลที่ได้ไปประมาณโครงการในส่วนของทรัพยากร ค่าใช้จ่าย และกำหนดการในการพัฒนา โดยใช้ข้อมูลจากอดีตที่มีความถูกต้อง การประมาณทำให้เกิดความเสี่ยงเนื่องจากปัจจัยต่างๆดังแสดงในรูปที่ 3.8 ดังนี้

1. ความซับซ้อนของโครงการ
2. ขนาดของโครงการ
3. ดัชนีความเป็นโครงสร้างของโครงการ

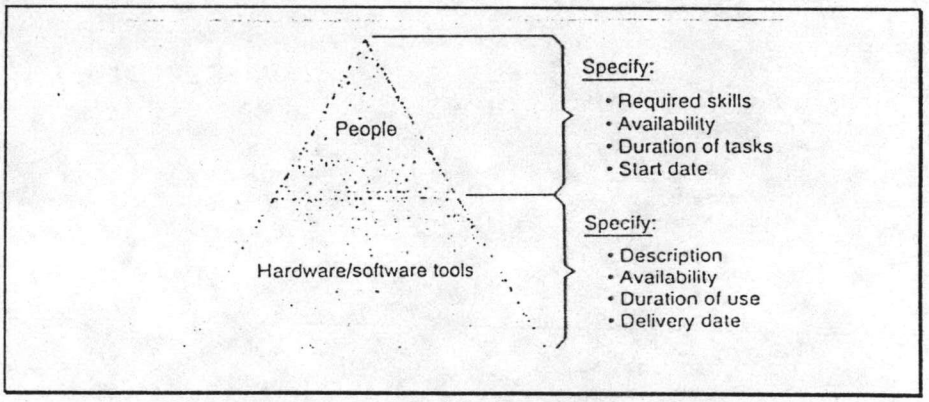


รูปที่ 3.8 ความสัมพันธ์ของการประมาณกับความเสี่ยง

ขั้นตอนของการวางแผนโครงการเริ่มตั้งแต่การกำหนดขอบเขต หน้าที่ ประสิทธิภาพ ข้อจำกัด ตัวประสาน และความเชื่อมั่น ของระบบที่ชัดเจนเข้าใจง่ายทั้งในระดับของฝ่ายบริหารและฝ่ายเทคนิค นอกจากนี้ยังต้องพิจารณาในปัจจัยอื่นที่เกี่ยวข้อง เช่น

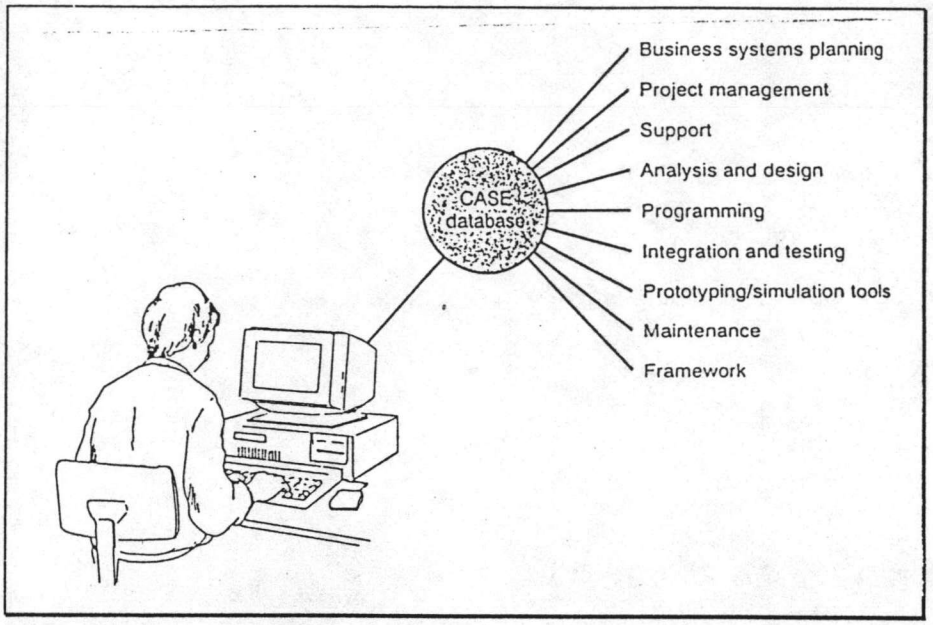
1. ฮาร์ดแวร์ที่ใช้ในการทำงาน
2. ซอฟต์แวร์เดิมที่มีอยู่แล้ว
3. คนที่เป็นผู้ใช้งานซอฟต์แวร์
4. กระบวนการทำงานของผู้ควบคุมเครื่อง

เมื่อกำหนดขอบเขตของระบบเสร็จแล้วจะทำการประมาณทรัพยากรที่ต้องใช้ในการพัฒนาระบบ ประกอบด้วยส่วนต่างๆดังแสดงในรูปที่ 3.9



รูปที่ 3.9 ทรัพยากรที่ใช้ในการพัฒนาซอฟต์แวร์

1. ทรัพยากรบุคคล เมื่อกำหนดขอบเขตเสร็จแล้ว ทำให้ทราบถึงบุคคลที่มีความสามารถเฉพาะด้านและจำนวนที่ต้องใช้
2. ทรัพยากรฮาร์ดแวร์ ประกอบด้วยส่วนต่างๆคือ ระบบและอุปกรณ์ที่ใช้ในการพัฒนา ระบบที่ใช้งานจริง
3. ทรัพยากรซอฟต์แวร์ เป็นเครื่องมือเพื่อช่วยในการพัฒนาซอฟต์แวร์ให้สะดวกและรวดเร็ว
4. เครื่องมือช่วยในการวางแผนทางธุรกิจ เป็นเครื่องมือง่ายๆแต่มีความสำคัญในด้านของ การบริหารโครงการ การควบคุมและจัดการข้อมูลที่ใช้ในการพัฒนา การวิเคราะห์ การออกแบบและการเขียนโปรแกรม การรวบรวมและทดสอบ การทำต้นแบบและแบบจำลอง และการบำรุงรักษา ดังแสดงในรูปที่ 3.10



รูปที่ 3.10 ระบบคอมพิวเตอร์เพื่อช่วยในงานวิศวกรรมซอฟต์แวร์

5. การนำกลับมาใช้ใหม่ โดยออกแบบซอฟต์แวร์ให้เป็นมอดูลต่างๆ เพื่อนำกลับมาใช้ใหม่ได้ช่วยให้คุณภาพสูงขึ้นและระยะเวลาในการพัฒนาสั้นลง โดยมีข้อพิจารณาดังนี้
   - ถ้าซอฟต์แวร์ที่มีอยู่แล้วตรงกับความต้องการสามารถนำมาใช้ได้เลย ค่าใช้จ่ายที่เกิดขึ้นมักจะต่ำกว่าการพัฒนาใหม่
   - ถ้าซอฟต์แวร์ที่มีอยู่แล้วต้องการการปรับปรุงบางส่วนเพื่อให้ใช้งานได้ ต้องพิจารณาให้รอบคอบ ค่าใช้จ่ายในการปรับปรุงอาจจะสูงกว่าการพัฒนาขึ้นมาใหม่

ในอดีตค่าใช้จ่ายของซอฟต์แวร์มีสัดส่วนต่ำกว่าฮาร์ดแวร์มาก ความผิดพลาดจากการประมาณมีผลกระทบน้อย แต่ในปัจจุบันซอฟต์แวร์กลับมีค่าใช้จ่ายสูงกว่า ทำให้การประมาณซอฟต์แวร์มีความสำคัญ แต่ยังไม่มีวิธีการวางแผนที่แน่นอน เนื่องจากมีปัจจัยเกี่ยวข้องจำนวนมากเช่น คน เทคนิค สิ่งแวดล้อม และการเมือง เป็นต้น อย่างไรก็ตามได้มีการศึกษาเพื่อหาขั้นตอนในการประมาณอย่างเป็นระบบและอยู่ในระดับที่ยอมรับได้

หลักในการประมาณค่าใช้จ่ายและระยะเวลาของพัฒนาซอฟต์แวร์ประกอบด้วย

1. เลื่อนการประมาณมาเป็นช่วงสุดท้ายของโครงการ
2. ใช้เทคนิคง่ายๆในการแบ่งเป็นโครงการย่อย แล้วแยกประมาณในแต่ละส่วนย่อย
3. พัฒนาแบบจำลองที่ซับซ้อนในการประมาณ

4. สร้างเครื่องมืออัตโนมัติสำหรับช่วยในการประมาณ

วิธีการแรกไม่สามารถทำได้ในทางปฏิบัติ ส่วนหลักการในข้ออื่นได้มีการศึกษากันอย่างกว้างขวางในทางวิศวกรรมซอฟต์แวร์และเปรียบเทียบความถูกต้องระหว่างกัน ในโครงการที่มีความซับซ้อนมักจะมีการแบ่งออกเป็นโครงการย่อยเพื่อให้การบริหารและแก้ปัญหาได้สะดวก แล้วนำมารวมในโครงการใหญ่เมื่อทำเสร็จ

วิธีการประมาณจำนวนบรรทัดและฟังก์ชันพอยท์

การนำเอาจำนวนบรรทัดและค่าฟังก์ชันพอยท์มาใช้ประมาณโครงการ โดยใช้เป็น

1. ตัวแปรที่ใช้ในการประมาณขนาดของแต่ละส่วนในซอฟต์แวร์
2. ฐานของเมตริกจากข้อมูลในอดีตเพื่อใช้ในการประมาณโครงการใหม่ที่จะทำ

ทั้งสองวิธีเป็นเทคนิคที่ต่างกันแต่มีลักษณะที่เหมือนกันคือ ผู้วางแผนโครงการเริ่มต้นจากการกำหนดขอบเขตของโครงการ แบ่งออกเป็นส่วนย่อย ทำการประมาณส่วนย่อยนี้แยกกัน และนำมารวมกันเพื่อประมาณโครงการทั้งหมด

เมื่อประมาณค่าจำนวนบรรทัดหรือค่าฟังก์ชันพอยท์ของซอฟต์แวร์เสร็จแล้ว นำมาใช้ร่วมกับข้อมูลกำลังการผลิต ซึ่งทำได้ 2 แนวทางคือ

1. ค่าที่ได้จากการประมาณแต่ละโครงการย่อย นำมาคำนวณกับอัตราการผลิตจะได้ค่าระยะเวลาของในการพัฒนาของส่วนย่อยนั้น
2. ค่าที่ได้จากการประมาณแต่ละโครงการย่อย นำมาคำนวณกับอัตราการผลิตซึ่งมีการปรับค่าตามระดับความซับซ้อนของแต่ละส่วนย่อยนั้น

วิธีการประมาณความพยายาม

การประมาณความพยายามของโครงการเป็นเทคนิคพื้นฐานในการพัฒนาโครงการทางวิศวกรรมเพื่อหาจำนวนคนและค่าใช้จ่ายที่ต้องใช้ในแต่ละช่วงเวลา โดยพิจารณาในแต่ละช่วงของวัฏจักรซอฟต์แวร์ ตั้งแต่การกำหนดขอบเขต การวิเคราะห์ การออกแบบ การเขียนโปรแกรม และการทดสอบงานแต่ละอย่างที่สัมพันธ์กัน ดังแสดงในตารางรูปที่ 3.11

Tasks / Functions	Requirements analysis	Design	Code	Test	Total
UICF	1.0	2.0	0.5	3.5	7
2DGA	2.0	10.0	4.5	9.5	26
3DGA	2.5	12.0	6.0	11.0	31.5
DSM	2.0	6.0	3.0	4.0	15
CGDF	1.5	11.0	4.0	10.5	27
PCF	1.5	6	3.5	5	16
DAM	4	14	5	7	30
Total*	14.5	61	26.5	50.5	152.5
Rate ($)	5200	4800	4250	4500	
Cost ($)	75,400	292,800	112,625	227,250	708,075

Estimated effort for all tasks

Estimated cost for all tasks

*All estimates are in person–months except where otherwise noted.

รูปที่ 3.11 ตารางเมตริกความพยายามในการพัฒนา

ข้อมูลของการประมาณมีความผิดพลาดเมื่อข้อตกลงที่ใช้ไม่มีคุณภาพ เนื่องจาก

1. การกำหนดขอบเขตของโครงการไม่ชัดเจน ทำให้ผู้วางแผนแปลความหมายผิด
2. ค่าอัตราการผลิตที่ใช้ในการคำนวณไม่เหมาะสม เนื่องจากเป็นข้อมูลที่เก่ามาก หรือคำนวณผิดพลาด ผู้วางแผนต้องหาสาเหตุ ทำการแก้ไข และวางแผนใหม่
3. การประมาณโดยใช้แบบจำลองทางคณิตศาสตร์ โดยสร้างสูตรมาช่วยในการพยากรณ์ ซึ่งเก็บข้อมูลมาจากโครงการตัวอย่างไม่มากนัก ทำให้แบบจำลองไม่เหมาะสมกับการพัฒนาในทุกสภาพแวดล้อม

แบบจำลองทรัพยากร (Resource Model) สร้างสมการคณิตศาสตร์สำหรับประมาณความพยายาม และระยะเวลาโครงการ โดยแบ่งออกเป็น 4 ประเภทตามจำนวนตัวแปรที่ใช้ในการคำนวณ

แบบจำลองโคโคโม (COCOMO Consturctive Cost Model) แบ่งการประมาณซอฟต์แวร์ออกเป็นลำดับชั้นต่างๆดังนี้

1. ระดับพื้นฐาน เป็นแบบจำลองที่ใช้ตัวแปรคงที่ตัวเดียวในการคำนวณความพยายามและค่าใช้จ่าย โดยเป็นฟังก์ชันที่ขึ้นอยู่กับจำนวนบรรทัดของโปรแกรมที่ประมาณ

2. ระดับกลาง เป็นการประมาณความพยายามในการพัฒนาซอฟต์แวร์ ที่ขึ้นอยู่กับขนาดของโปรแกรมและค่าใช้จ่ายที่ใช้โดยประเมินจาก ผลิตภัณฑ์ ฮาร์ดแวร์ คน และลักษณะของโครงการ
3. ระดับสูง เป็นการรวมเอาลักษณะทุกอย่างของการประมาณในระดับกลาง และประเมินค่าใช้จ่ายในทุกขั้นตอนของการพัฒนา ในกระบวนการวิศวกรรมซอฟต์แวร์

แบบจำลองนี้แบ่งโครงการซอฟต์แวร์ออกเป็น 3 ระดับคือ

1. โครงการขนาดเล็ก ง่ายๆ ใช้จำนวนคนไม่มากที่มีความเชี่ยวชาญในด้านที่ต้องการอยู่แล้ว และความต้องการไม่ซับซ้อน
2. โครงการขนาดกลาง เป็นโครงการซอฟต์แวร์ที่ต้องใช้คนที่มีความชำนาญหลายด้าน ความต้องการผสมกันระหว่างซับซ้อนและธรรมดา
3. โครงการขนาดใหญ่ มีข้อจำกัดมากทั้งทางด้านซอฟต์แวร์ ฮาร์ดแวร์ และการทำงาน

ในแบบจำลองระดับพื้นฐานได้พัฒนาปัจจัยที่มีผลต่อค่าใช้จ่ายออกเป็น 4 กลุ่มดังนี้

1. ลักษณะของผลิตภัณฑ์
2. ลักษณะของฮาร์ดแวร์
3. ลักษณะของบุคคล
4. ลักษณะของโครงการ

นำค่าปัจจัยที่เกี่ยวข้องนี้มาจัดลำดับและให้คะแนนของแต่ละปัจจัย เพื่อนำไปปรับกับค่าความพยายามที่ได้มาในตอนแรก

เครื่องมือที่ช่วยในการประมาณอัตโนมัติ เทคนิคที่ใช้ในการแบ่งโครงการเป็นส่วนย่อย และแบบจำลองที่ใช้ในการคำนวณสามารถนำมาพัฒนาเป็นซอฟต์แวร์สำเร็จรูป เพื่อเป็นเครื่องมือช่วยผู้วางแผนในการประมาณค่าใช้จ่ายและความพยายาม แม้ว่าจะมีเครื่องมือแบบนี้จำนวนมากแต่ก็มีลักษณะคล้ายกันและต้องการข้อมูลต่างๆดังนี้

1. ข้อมูลที่ใช้ในการประมาณขนาดโครงการซอฟต์แวร์เช่น จำนวนบรรทัด หรือฟังก์ชันพอยท์
2. ข้อมูลที่ใช้ในการกำหนดลักษณะของซอฟต์แวร์ เช่น ความซับซ้อน ระดับความเชื่อมั่น และระดับจุดวิกฤตที่ยอมรับ
3. รายละเอียดของทีมงานที่พัฒนาและสภาพแวดล้อม

จากข้อมูลต่างๆนี้ มีการพัฒนาแบบจำลองสำหรับเป็นเครื่องมือช่วยในการประมาณความพยายาม ค่าใช้จ่าย บุคคล ปริมาณงาน การจัดลำดับงาน และการวิเคราะห์ความเสี่ยง ตัวอย่างซอฟต์แวร์เครื่องมือช่วยในการพัฒนาซอฟต์แวร์ในท้องตลาด มีดังนี้

บีวายแอล (BYL Before You Leap) พัฒนาโดยบริษัทกอร์ดอนกรุ๊ป(Gordon Group) วิโคโม (WICOMO Wang Institues Cost Model) ของสถาบันแวงก์ (Wang Instutute) และ เด็คแพลน (DECPlan) ของบริษัทดิจิตอล (Digital Equipment Corporation) ซอฟต์แวร์เหล่านี้พัฒนาโดยใช้พื้นฐานจากแบบจำลองของโคโคโม ต้องการข้อมูลจำนวนบรรทัด และสามารถระบุภาษาที่ใช้ในการพัฒนาได้ ผลลัพธ์ที่ได้คือ ข้อมูลการประมาณระยะเวลาโครงการ ความพยายาม ค่าเฉลี่ยของจำนวนคณะทำงานที่ใช้ และค่าเฉลี่ยของกำลังการผลิต ข้อมูลเหล่านี้นำไปใช้ได้ทั้งในโครงการย่อยหรือในโครงการทั้งหมด

สลิม(SLIM) ใช้สำหรับประมาณค่าใช้จ่ายโครงการ โดยใช้วัฏจักรซอฟต์แวร์ของเรย์ลายและนอร์เดอร์(Rayleigh-Norder) และแบบจำลองการประมาณของพุทนาม(Putnam Estimation) มีการใช้เทคนิคโปรแกรมเชิงเส้น การจำลองทางสถิติ การประเมินซอฟต์แวร์ และการจัดลำดับงานในการประมาณโครงการซอฟต์แวร์

เอสทิแมคส์(ESTIMACS) ใช้แบบจำลองการประมาณมาโคร(Macro Estimation Model) ที่พัฒนามาจากวิธีฟังก์ชันพอยท์ ให้ข้อมูลของ ความพยายามในการพัฒนาระบบ คณะทำงานและค่าใช้จ่าย โครงแบบของฮาร์ดแวร์ และความเสี่ยง

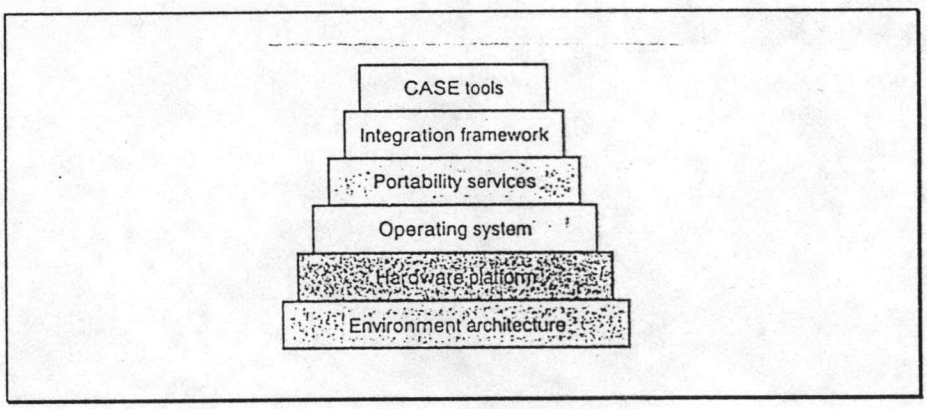
<u>การใช้ระบบคอมพิวเตอร์ช่วยในการพัฒนาซอฟต์แวร์</u>

ระบบคอมพิวเตอร์สำหรับช่วยในงานวิศวกรรมซอฟต์แวร์ (CASE - Computer-Aided Software Engineering) มีพัฒนาในสาขาต่างๆ ตั้งแต่เครื่องมือง่ายๆเพื่อช่วยในงานเฉพาะด้าน ไปจนถึงระบบที่ซับซ้อนครอบคลุมทุกอย่างในการพัฒนาดังแสดงในรูปที่ 3.12

1. เครื่องมือช่วยในการวางแผนทางธุรกิจ
2. เครื่องมือในการบริหารโครงการ ผู้จัดการโครงการต้องมีการประมาณ การควบคุม และการติดตามความคืบหน้า ประกอบด้วย
   - เครื่องมือในการวางแผนโครงการ
   - เครื่องมือในการบันทึกความต้องการ
   - เครื่องมือสำหรับเมตริกและการบริหาร
3. เครื่องมือในการสนับสนุนโครงการ
   - เครื่องมือในการทำเอกสาร
   - เครื่องมือสำหรับซอฟต์แวร์ระบบ

- เครื่องมือสำหรับการประกันคุณภาพ
- เครื่องมือในการบริหารฐานข้อมูล

4. เครื่องมือในการวิเคราะห์และออกแบบระบบ
   - เครื่องมือในการวิเคราะห์และออกแบบโครงสร้างระบบ
   - เครื่องมือในการทำต้นแบบและแบบจำลอง
   - เครื่องมือในการออกแบบตัวประสานและการพัฒนา
   - เครื่องจักรในการวิเคราะห์และออกแบบ
5. เครื่องมือในการเขียนโปรแกรม
   - เครื่องมือในการเขียนโปรแกรมทั่วไป
   - เครื่องมือในการเขียนโปรแกรมในยุคที่สี่
   - เครื่องมือในการเขียนโปรแกรมเชิงวัตถุ
6. เครื่องมือในการรวมและทดสอบซอฟต์แวร์
7. เครื่องมือในการทำต้นแบบ
8. เครื่องมือในการบำรุงรักษาซอฟต์แวร์



รูปที่ 3.12 ลำดับชั้นของเคส

ส่วนที่ 2

ข้อกำหนดและการใช้งานโปรแกรมที่พัฒนา