Software cost estimation

Week 14

ดัดแปลงจาก slides ของหนังสือ Software Engineering [1]

หัวข้อที่จะศึกษา

- Software productivity
- Estimation techniques
- Algorithmic cost modelling
- Project duration and staffing

Fundamental estimation questions

- สิ่งที่ต้องใช้ในการสร้างซอฟต์แวร์
  - ความพยายามมากน้อยแค่ไหน
  - เวลาตามปฏิทินที่ต้องใช้
- มีต้นทุนของกิจกรรมต่างๆ ซึ่งประกอบด้วยอะไรบ้าง
- โดยทั่วไปการประเมินโครงการและการวางแผนโครงการจะดำเนินสลับกันไป
  - การประเมินราคาต้องอาศัยประสบการณ์

Software cost components

- ต้นทุนด้าน Hardware และ software
- ต้นทุนด้านการเดินทางและการจัดอบรม
- ต้นทุนด้านค่าแรง (เป็นค่าใช้จ่ายส่วนใหญ่ของโครงการ)
  - เงินเดือนและค่าจ้างพนักงาน
  - ภาษีและค่าประกันสังคม
- ต้นทุนอื่นๆ ที่ต้องนำมาคิด (overhead)
  - อาคารสถานที่ ค่าสาธารณูปโภค
  - networking และ communications
- อื่นๆ เช่น วัสดุสำนักงาน ห้องสมุด อาหารเครื่องดื่ม สวัสดิการพนักงาน ฯลฯ

Costing and pricing

- การประมาณการเพื่อหาต้นทุนสำหรับนักพัฒนาในการผลิตระบบซอฟต์แวร์
  - ไม่มีความสัมพันธ์ที่ตรงไปตรงมาระหว่างต้นทุนการพัฒนากับราคาที่เรียกเก็บจากลูกค้า
- ควรนำปัจจัยรอบด้านมาพิจารณาเรียกเก็บค่าใช้จ่ายจากลูกค้า
  - ภาพกว้างขององค์กร (ของลูกค้า)
  - เศรษฐกิจ
  - การเมือง
  - สภาพทางธุรกิจ

Software pricing factors

- เงื่อนไขสัญญา (Contractual terms)
  - เพื่อลดราคาซอฟต์แวร์ ลูกค้าอาจเต็มใจให้นักพัฒนารักษาความเป็นเจ้าของของ source code และนำมา reuse ในโครงการอื่นได้
- ความผันผวนของการประมาณต้นทุน (Cost estimate uncertainty)
  - หากไม่มั่นใจในการประเมินต้นทุน อาจขึ้นราคาเผื่อเหตุฉุกเฉินที่อาจเกิดขึ้น โดยยังคงมีผลตอบแทนที่เหมาะสม
- สุขภาพทางการเงิน (Financial health)
  - ถ้าบริษัทที่มีปัญหาทางการเงิน ก็อาจลดราคาลงเพื่อให้ได้สัญญา
  - กำไรน้อยกว่าปกติหรือแค่คุ้มทุน ดีกว่าต้องเลิกกิจการ
  - กระแสเงินสดย่อมสำคัญกว่ากำไรในยุคเศรษฐกิจที่ยากลำบาก

Software pricing factors

- โอกาสทางการตลาด (Market opportunity)
  - ผู้พัฒนาอาจเสนอราคาต่ำเพราะต้องการแย่งพื้นที่ในการเข้าสู่ตลาดซอฟต์แวร์
  - ในระยะเริ่มต้นอาจจะยอมรับกำไรที่ต่ำ แต่การได้ทำโครงการหนึ่งอาจทำให้องค์กรมีโอกาสทำกำไรมากขึ้นในโครงการต่อ ๆ ไป เนื่องจากการสั่งสมประสบการณ์
- ความต้องการที่ไม่ชัดเจน (Requirements volatility)
  - หากพบว่ามีแนวโน้มที่จะเปลี่ยนแปลง requirement เราอาจลดราคาลงบ้าง เพื่อให้ชนะการประมูลและได้สัญญา
  - หลังจากทำสัญญาแล้ว อาจเรียกราคาสำหรับ requirement ที่เปลี่ยนแปลงได้มากขึ้น

Software productivity

- Software productivity คือการวัดอัตราที่วิศวกรแต่ละรายสามารถผลิตซอฟต์แวร์และเอกสารที่เกี่ยวข้อง
- การวัดนี้จะไม่เน้นการวัดคุณภาพ (แม้ว่าการประกันคุณภาพจะเป็นปัจจัยในการประเมินผลิตภาพ)
- โดยปกติ จะเป็นการวัดผลการทำงานที่มีประโยชน์ เทียบกับหน่วยเวลา

Productivity measures

- การวัดเชิงขนาด (size related measured)
  - วัดผลลัพธ์จากกระบวนการซอฟต์แวร์ เช่น จำนวนบรรทัดของซอร์สโค้ดที่ส่งมอบ จำนวนของคำสั่งที่เป็น object code ฯลฯ
- การวัดเชิงฟังก์ชัน (function-related measure)
  - วัดจากการประมาณการของฟังก์ชันการทำงานของซอฟต์แวร์ที่ส่งมอบ การวัดประเภทนี้จะรู้จักในรูปแบบ function-points

Measurement problems

- การประมาณขนาดของหน่วยวัด (เช่น จำนวน function points)
- การประมาณโปรแกรมเมอร์ทั้งหมดที่ใช้ (คน/เดือน)
- การประมาณประสิทธิภาพการทำงานของทีมงาน (เช่น ทีมเอกสาร)

Lines of code

- จำนวนบรรทัดของ code คืออะไร?
  - แนวคิดนี้ถูกเสนอครั้งแรกเมื่อโปรแกรมถูกพิมพ์ลงบนการ์ดที่มีหนึ่งบรรทัดต่อการ์ด
  - สอดคล้องกับภาษา java ที่แต่ละคำสั่งสามารถมีได้หลายบรรทัดหรือมีหลายคำสั่งในหนึ่งบรรทัด
- โมเดลนี้ถือว่ามีความสัมพันธ์เชิงเส้นระหว่างขนาดระบบและปริมาณของเอกสารประกอบ

การนับจำนวนบรรทัดของ code

- นับ
  - บรรทัดที่เป็น Source Code
  - บรรทัดที่พัฒนาโดยบุคลากร
  - ส่วนของการประกาศค่า (Declaration) ที่เป็นส่วนของ Instruction
- ไม่นับ
  - ส่วนของการทดสอบ (Test Driver)
  - ส่วนงานที่รองรับการทำงานอื่นๆ
  - ส่วนของการขยายความ
  - Comment

Productivity comparisons

- โปรแกรมเมอร์ที่ใช้ภาษาระดับต่ำ (low level language) จะมี productive สูงกว่า
  - Code ที่มีการทำงานอย่างเดียวกัน ในภาษาระดับต่ำ จะมีจำนวนบรรทัดที่มากกว่าภาษาระดับสูง
- โปรแกรมเมอร์ที่ทำงานละเอียด จะมี productivity สูงกว่า
  - การวัด productivity โดยอาศัย LoC (lines of code) นั้น พบว่าโปรแกรมเมอร์ที่เขียน code ที่เปิดเผยรายละเอียด (verbose code) จะมี productive มากกว่าโปรแกรมเมอร์ที่เขียน compact code

System development times

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	4 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

Function points

- วัดจากการรวมกันของคุณสมบัติของโปรแกรม เช่น
  - inputs และ outputs
  - การโต้ตอบกับ user
  - การ interfaces กับภายนอก
  - files ที่ถูกใช้โดยระบบ
- น้ำหนักที่ให้กับแต่ละคุณสมบัติอาจแตกต่างกันไป ดังนั้นค่า function point จึงได้จากการคูณจำนวนและน้ำหนักของคุณสมบัติแต่ละประเภท

$$UFC = \sum(\text{number of elements of given type}) \times (\text{weight})$$

UFP (Unjustified Function Point) คือ function point ที่ยังไม่คิดตัวแปรอื่นๆ ที่ส่งผลกระทบ

Function points

- การนับ function point อาจได้รับการปรับปรุงตามความซับซ้อนของโครงการ
- FPs อาจนำไปใช้กับการประเมิน LOC โดยขึ้นอยู่กับค่าเฉลี่ยของ LOC ต่อ FP ในภาษาโปรแกรมที่กำหนด
  - LOC = AVC * number of function points;
  - เมื่อ AVC เป็นค่า factor เฉลี่ยที่ขึ้นกับภาษาที่ใช้ในการพัฒนา เช่น 200-300 สำหรับ assemble language จนถึง 2-40 สำหรับภาษา 4GL;
- FPs เป็นเรื่องละเอียดอ่อน การประเมินขึ้นอยู่กับประสบการณ์ของผู้ประเมิน
  - ไม่สามารถใช้ระบบอัตโนมัติช่วยประเมินสิ่งนี้ได้

Value Adjustment Factor (VAF)

1. Data communication (ข้อมูลและการควบคุมข้อมูลที่ใช้ในซอฟต์แวร์)
2. Distributed function (การประมวลผลข้อมูลแบบกระจาย)
3. Performance (ประสิทธิภาพด้านต่างๆของซอฟต์แวร์)
4. Heavily used configuration (โปรแกรมที่ต้องมีการคอนฟิกบ่อยครั้ง)
5. Transaction rates (อัตราการทำงานของข้อมูล)
6. On-line data entry (สามารถที่จะควบคุมข้อมูลแบบออนไลน์)
7. Design for end-user efficiency (ประสิทธิภาพในการออกแบบสำหรับผู้ใช้)

Value Adjustment Factor (VAF)

8. On-line update (คุณสมบัติในการปรับปรุงข้อมูลแบบออนไลน์)
9. Complex processing (ความซับซ้อนในการประมวลผล)
10. Usability in other applications (มีความยืดหยุ่นที่จะใช้ร่วมกับแอปพลิเคชันอื่น)
11. Installation ease (ความง่ายในการติดตั้ง)
12. Operational ease (ความง่ายในการใช้งาน)
13. Multiple sites (มีการแบ่งการทำงานเป็นหลายที่หรือไม่)
14. Facilitate change (สามารถที่เปลี่ยนแปลงซอฟต์แวร์ได้ง่าย)

Object points

- Object points (หรือบางที่ก็เรียก application points) เป็นการวัดคล้ายๆ กับ function point
- Object points ไม่ใช่ object ในเรื่อง classes
- จำนวนของ object points ในโปรแกรมสามารถประเมินได้จากอะไรบ้าง
  - จำนวนของหน้าจอ (separate screens)
  - จำนวนรายงานที่สร้างได้จากระบบ
  - จำนวนโมดูลที่ต้องสร้าง หรือจำนวนฐานข้อมูลที่ต้องใช้

Object point estimation

- Object points สามารถประเมินจาก specification ได้ง่ายกว่า function points
  - ดูได้จากจำนวน screens, reports และ programming language modules
- สามารถใช้ประเมินราคาได้ตั้งแต่ช่วงต้นของการพัฒนา
  - ซึ่งแน่นอนว่า ไม่สามารถประเมินจำนวนบรรทัดของ code ที่ใกล้เคียงได้เลย

Productivity estimates

- การประเมิน LOC โดยทั่วไป
  - Real-time embedded systems, 40-160 LOC/P ต่อเดือน
  - Systems programs , 150-400 LOC/P ต่อเดือน
  - Commercial applications, 200-900 LOC/P ต่อเดือน
- ใน object points นั้น productivity สามารถวัดได้ในช่วง 4 ถึง 50 object points ต่อเดือน ขึ้นอยู่กับเครื่องมือและความสามารถของนักพัฒนา

Factors affecting productivity

- ประสบการณ์ (Application domain experience)
  - ความรู้เกี่ยวกับโดเมนแอปพลิเคชันเป็นสิ่งจำเป็นสำหรับการพัฒนาซอฟต์แวร์ที่มีประสิทธิภาพ วิศวกรที่เข้าใจโดเมนอยู่แล้วมักจะมี productivity สูงสุด
- คุณภาพของกระบวนการ (Process quality)
  - กระบวนการพัฒนาที่ใช้สามารถมีผลกระทบอย่างมีนัยสำคัญต่อผลผลิต
- ขนาดโครงการ (Project size)
  - โครงการใหญ่ ต้องใช้เวลาในการสื่อสารในทีมมากขึ้น มีเวลาในการพัฒนาน้อยลง ดังนั้นผลิตภาพของแต่ละบุคคลจึงลดลง

Factors affecting productivity

- การสนับสนุนด้านเทคโนโลยี (Technology support)
  - เทคโนโลยีการสนับสนุนที่ดี เช่น CASE Tool ระบบการจัดการการกำหนดค่า ฯลฯ สามารถเพิ่ม productivity ได้
- สภาพแวดล้อมในการทำงาน (Working environment)
  - สภาพแวดล้อมการทำงานที่เงียบสงบพร้อมพื้นที่ทำงานส่วนตัวมีส่วนช่วยในการเพิ่มผลผลิต

Quality and productivity

- ตัววัดทางด้านปริมาณทั้งหมดที่กล่าวมา มักจะไม่สะท้อน productivity เนื่องจากไม่ได้คำนึงถึง**คุณภาพ**
- Productivity โดยทั่วไปสามารถเพิ่มขึ้นด้วยการเพิ่มต้นทุนด้านคุณภาพ
- ยังไม่มีใครสามารถบอกได้อย่างชัดเจนว่าตัวชี้วัดประสิทธิภาพ/คุณภาพมีความเกี่ยวข้องกันอย่างไร
- ถ้า requirement มีการเปลี่ยนแปลงตลอดเวลา วิธีการนับ line of code จะไม่มีความหมายเนื่องจากตัวโปรแกรมไม่คงที่

Estimation techniques

- ไม่มีวิธีการอย่างง่าย ๆ ในการประเมินสิ่งที่จำเป็นในการพัฒนาระบบซอฟต์แวร์ได้อย่างแม่นยำ
  - requirement ที่ไม่ชัดเจนเพียงพอจากผู้ใช้
  - ซอฟต์แวร์อาจถูกนำไปใช้งานบนคอมพิวเตอร์ที่ไม่คุ้นเคยหรือใช้เทคโนโลยีใหม่
  - ไม่รู้ศักยภาพของคนที่ทำงานในโครงการ
- การประมาณการต้นทุนโครงการอาจเริ่มจากฝ่ายนักพัฒนา
  - การประมาณการจะเป็นตัวกำหนดงบประมาณ งบประมาณจะถูกนำไปกำหนดผลิตภัณฑ์

Changing technologies

- การเปลี่ยนแปลงเทคโนโลยีอาจทำให้ประสบการณ์การประเมินไม่สามารถใช้ได้ผลกับโครงการใหม่ๆ
  - มีการประมวลผลแบบกระจายมากกว่าการรวมศูนย์
  - มีการใช้บริการเว็บกว้างขวางมากขึ้น
  - การใช้ ERP หรือระบบฐานข้อมูลเป็นศูนย์กลาง
  - การใช้ซอฟต์แวร์ที่มีจำหน่ายทั่วไป
  - การพัฒนาโดยการนำกลับมาใช้ใหม่
  - การพัฒนาโดยใช้ภาษาสคริปต์
  - การใช้ CASE tool และเครื่องมือช่วยสร้างโปรแกรม

Estimation techniques

- Algorithmic cost modelling.
- Expert judgement.
- Estimation by analogy.
- Parkinson's Law.
- Pricing to win.

Estimation techniques

- การใช้แบบจำลองอัลกอริทึมเป็นฐานในการประเมิน (Algorithmic cost modelling)
  - การสร้างแบบจำลองขึ้นมาจากประสบการณ์ในอดีต
  - นำเมตริกซอฟต์แวร์ (โดยปกติดูจากขนาด) กับต้นทุนโครงการมาเป็นแนวทาง
  - ทำค่าประมาณขึ้นจากตัวชี้วัดนั้น
- การประเมินโดยผู้เชี่ยวชาญ (Expert judgement)
  - ขอคำปรึกษาจากผู้เชี่ยวชาญหลายคนเกี่ยวกับเทคนิคการพัฒนาซอฟต์แวร์และโดเมนแอปพลิเคชัน
  - ผู้เชี่ยวชาญแต่ละคนประเมินต้นทุนโครงการ
  - นำค่าประมาณการมาเปรียบเทียบและอภิปราย
  - กระบวนการประมาณการจะถูกทำซ้ำจนกว่าจะได้ประมาณการตามที่ตกลงกันไว้

Estimation techniques

- การเทียบเคียงกับระบบที่เคยพัฒนามาแล้ว (Estimation by analogy)
  - เทคนิคนี้อาศัยการเปรียบเทียบกับโครงการอื่น (ในโดเมนแอปพลิเคชันเดียวกัน) ที่เสร็จสมบูรณ์แล้ว
  - ค่าใช้จ่ายของโครงการใหม่จะถูกประมาณการโดยเปรียบเทียบกับโครงการที่เสร็จสมบูรณ์เหล่านั้น
- Parkinson's Law
  - เป็นวิธีการที่จะพัฒนาซอฟต์แวร์ภายใต้ระยะเวลาและทรัพยากรบุคคลที่จำกัด
  - ต้นทุนจะถูกกำหนดโดยทรัพยากรมากกว่าการประเมินตามวัตถุประสงค์
  - หากต้องส่งมอบซอฟต์แวร์ภายใน 12 เดือนและมีนักพัฒนา 5 คน จะสามารถประเมินความพยายามที่ต้องใช้อยู่ที่ประมาณ 60 คนต่อเดือน

Estimation techniques

- ประเมินจากราคาที่จะทำให้ได้งาน (Pricing to win)
  - ต้นทุนซอฟต์แวร์ถูกประมาณการจากประเมินเงินที่ลูกค้ามีจ่ายสำหรับโครงการ
  - ความพยายามในการพัฒนา (โดยประมาณ) ขึ้นอยู่กับงบประมาณของลูกค้า ไม่ได้ขึ้นอยู่กับฟังก์ชันของซอฟต์แวร์ (ลงแรงตามงบประมาณที่ลูกค้ายินดีจ่าย)
- COCOMO II
  - โมเดลในการประเมินราคาซอฟต์แวร์ หรือ Software Costing Model ซึ่งโมเดลนี้ถูกสร้างขึ้น ในปี 1981
  - เป็นที่ยอมรับและนำเอาไปใช้กันแพร่หลายในสหรัฐอเมริกา
  - นำเอาความแตกต่างของแต่ละโครงการ, ลักษณะเฉพาะ, ผู้ที่เกี่ยวข้องต่างๆ มาคิดคำนวณค่าออกมาเป็นตัวเลข

Pricing to win

- กำหนดมูลค่าโครงการตามที่ลูกค้ายินดีจ่าย
- ข้อดี:
  - นักพัฒนาได้รับสัญญาจ้างงาน
- ข้อเสีย:
  - ความน่าจะเป็นที่ลูกค้าจะได้ระบบที่เขาต้องการนั้นมีน้อย
  - ต้นทุนไม่ได้สะท้อนถึงงานที่ต้องการอย่างแม่นยำ

Top-down and bottom-up estimation

- การประเมินอาจจะใช้วิธีจากบนลงล่างหรือล่างขึ้นบน
- จากบนลงล่าง
  - เริ่มที่ระดับระบบและประเมินฟังก์ชันการทำงานของระบบโดยรวม แล้วค่อยประเมินลงมาในระบบย่อย
- จากล่างขึ้นบน
  - เริ่มต้นที่ระดับส่วนประกอบและประเมินสิ่งที่ต้องทำในแต่ละส่วนประกอบ
  - ทำจนครบเพื่อให้ได้ค่าประมาณขั้นสุดท้าย

Top-down estimation

- สามารถใช้งานได้แม้ไม่มีความรู้เกี่ยวกับสถาปัตยกรรมระบบและส่วนประกอบที่อาจจะอยู่ในระบบนั้น
- สามารถพิจารณาต้นทุนต่าง ๆ อย่างรอบด้านเช่น integration, configuration management และ documentation.
- สามารถประเมินค่าใช้จ่ายในการแก้ไขปัญหาทางเทคนิคที่จำเป็นได้

Bottom-up estimation

- ใช้งานได้เมื่อทราบสถาปัตยกรรมของระบบและสามารถระบุส่วนประกอบต่าง ๆ อย่างครบถ้วน
- อาจเป็นวิธีที่เหมาะสม หากระบบได้รับการออกแบบมาอย่างละเอียด
- อาจประเมินต้นทุนของกิจกรรมหลาย ๆ ด้านได้ต่ำเกินไป เช่น การรวมระบบและเอกสารประกอบ

Estimation methods

- แต่ละวิธีมีจุดแข็งและจุดอ่อน
- การประมาณค่าดำเนินการการหลาย ๆ วิธี ขนานกันไป
  - ถ้าแต่ละวิธีให้ผลต่างกันมาก แสดงว่ามีข้อมูลไม่เพียงพอที่จะประมาณการ ควรดำเนินการบางอย่างเพื่อหาข้อมูลเพิ่มเติมเพื่อให้ประมาณการได้แม่นยำยิ่งขึ้น
  - สุดท้ายแล้ว การกำหนดราคาแบบ Pricing to win อาจเป็นวิธีเดียวที่ใช้ได้

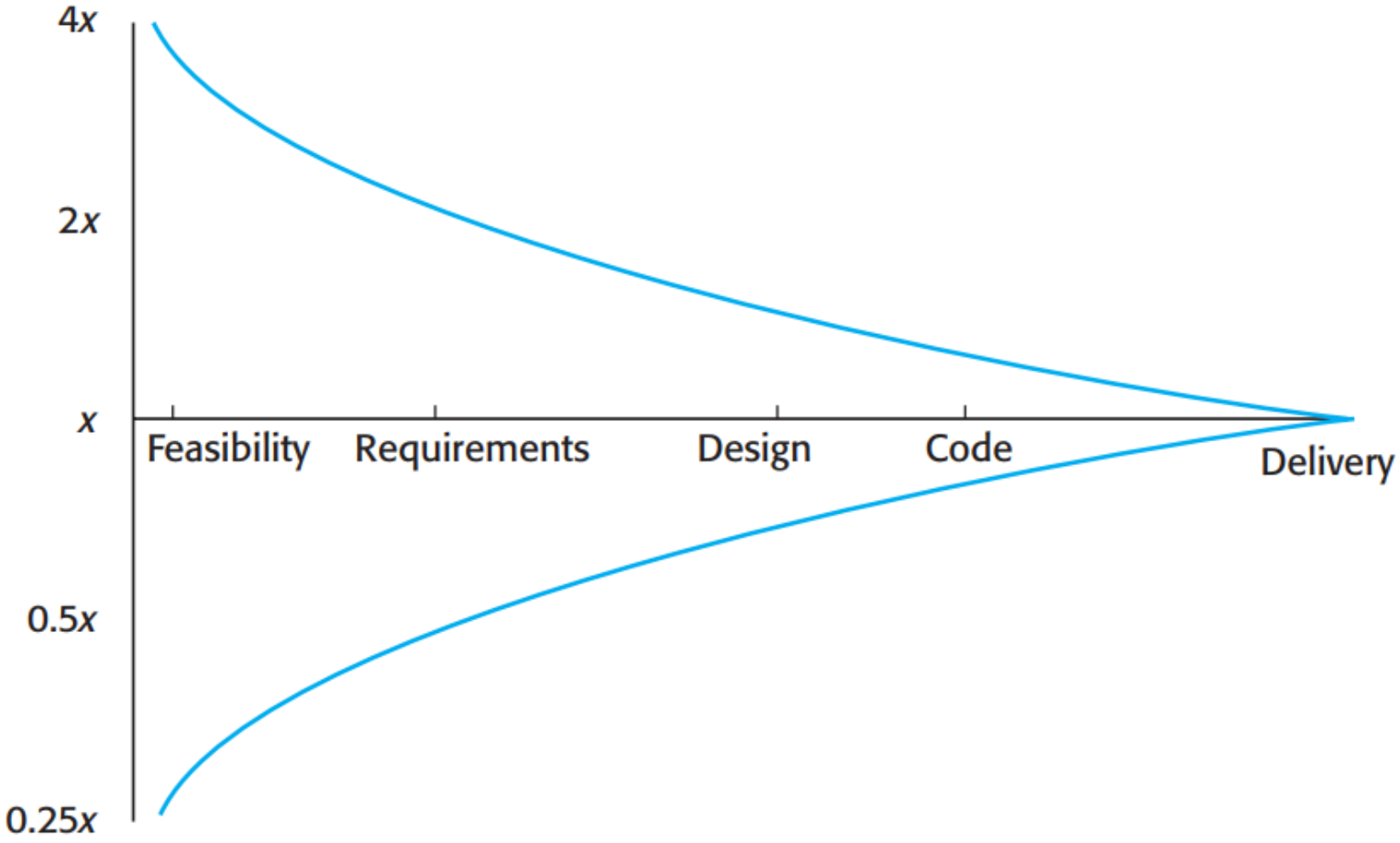
Pricing to win

- วิธีการตกลงราคานี้อาจดูเหมือนผิดจรรยาบรรณและไม่เป็นธุรกิจ
  - อย่างไรก็ตามเมื่อขาดข้อมูลรายละเอียด อาจเป็นเพียงกลยุทธ์เดียวที่เหมาะสม
- ต้นทุนโครงการจะถูกตกลงกันบนพื้นฐานของข้อเสนอโครงร่างและการพัฒนาถูกจำกัดด้วยต้นทุนนั้น
- อาจมีการเจรจาข้อกำหนด specification หรือแนวทาง evolution ที่ใช้สำหรับการพัฒนาระบบ

Estimation accuracy

- ขนาดที่แท้จริงของระบบซอฟต์แวร์สามารถทราบได้อย่างแม่นยำเมื่อเสร็จสิ้นแล้วเท่านั้น
- มีหลายปัจจัยที่ส่งผลต่อขนาดในขั้นสุดท้าย
  - การใช้ COTS และ components
  - ภาษาโปรแกรม
  - การกระจายของระบบ
- เมื่อกระบวนการพัฒนาดำเนินไปการประมาณขนาดจะแม่นยำยิ่งขึ้น

Estimate uncertainty



Cost Estimate

- Size
  - Line of Code, Function Points
- Effort
  - Manpower
- Time
  - Man Month (MM), Person Month (PM)
- Investment
  - Overhead (ค่าน้ำ ค่าไฟฟ้า ค่าธารณูปโภค ฯลฯ)

Effort Estimation

$$productivity = \frac{Line\ of\ Code\ \ or\ \ Function\ Point}{Effort\ (Man\ Month)}$$

Productivity = ค่าประสิทธิผลในการผลิต

Line of Code = จำนวนบรรทัดของซอฟต์แวร์

Function Point = ค่าจากการวัดฟังก์ชันพอยต์

Effort = กำลังคนหรือแรงงานที่ต้องใช้ในการพัฒนาซอฟต์แวร์

The COCOMO model (Constructive Cost Modeling)

- เป็นแบบจำลองเชิงประจักษ์ตามประสบการณ์ของโครงการ (ที่ได้ประสบการณ์จากการใช้งานจริง)
- เป็นโมเดลอิสระที่ได้รับการจัดทำเป็นเอกสารอย่างดี ซึ่งไม่ได้ผูกติดอยู่กับผู้จำหน่ายซอฟต์แวร์รายใดรายหนึ่ง
- มีประวัติศาสตร์อันยาวนานตั้งแต่รุ่นแรกที่ตีพิมพ์ในปี 1981 (COCOMO-81) ผ่านการพัฒนาจนถึง COCOMO II
- COCOMO II คำนึงถึงแนวทางต่าง ๆ ในการพัฒนาซอฟต์แวร์ เช่น rapid software development, Reuse, database programming ฯลฯ

COCOMO 81

Project complexity	Formula	Description
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

PM: the effort estimate in person-months.

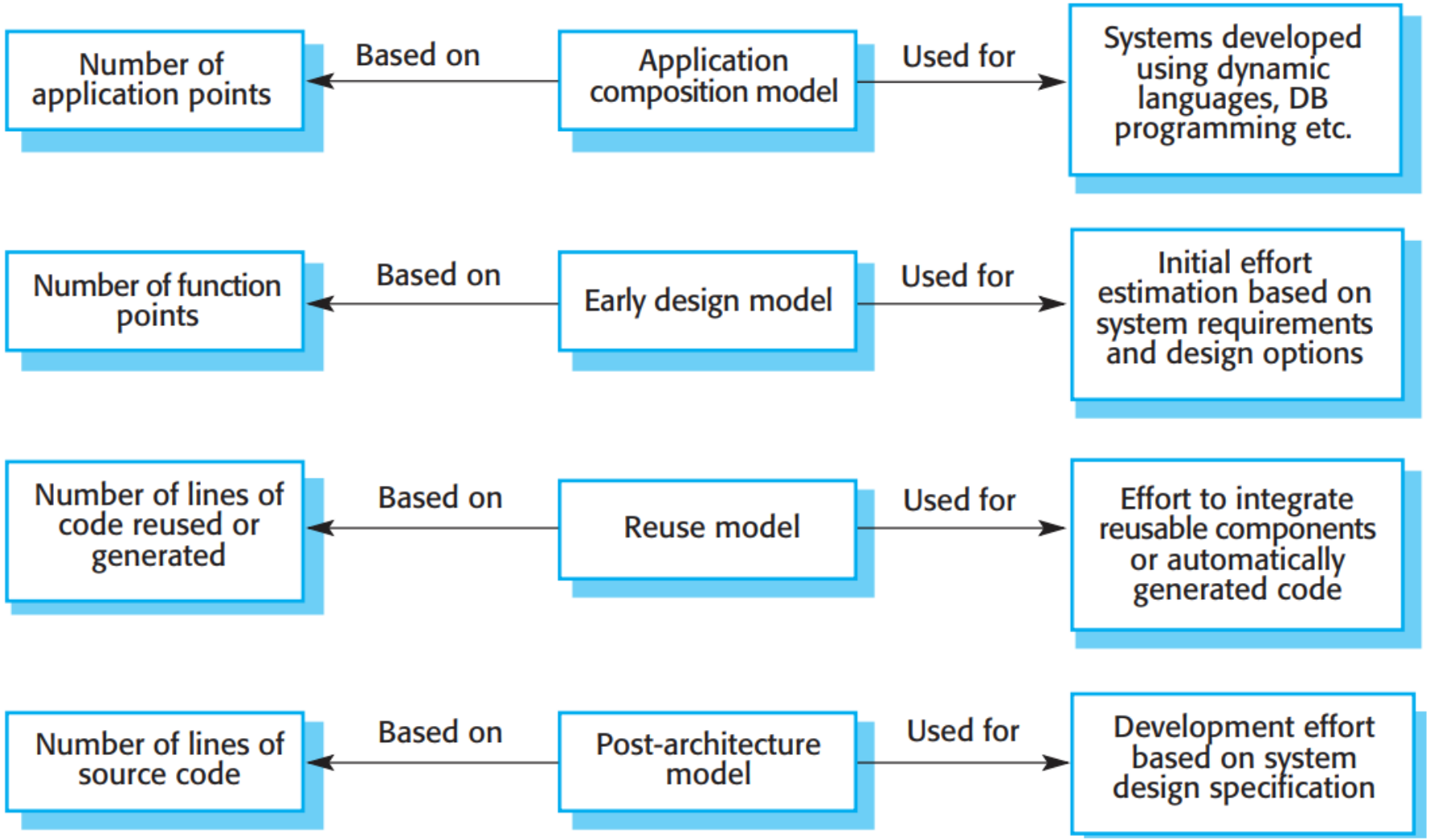
COCOMO II

- COCOMO 81 ได้รับการพัฒนาโดยเงื่อนไขตั้งต้นว่าจะใช้กระบวนการ waterfall และซอฟต์แวร์ทั้งหมดจะได้รับการพัฒนาตั้งแต่กระดาษเปล่า
- นับตั้งแต่เริ่มต้น COCOMO 81 มีการเปลี่ยนแปลงมากมายในแนวปฏิบัติด้านวิศวกรรมซอฟต์แวร์ และ COCOMO II ได้รับการออกแบบมาเพื่อรองรับแนวทางต่างๆ ในการพัฒนาซอฟต์แวร์ในปัจจุบัน

COCOMO 2 models

- COCOMO 2 ได้รวมเอาโมเดลย่อยต่างๆ ที่ช่วยให้สามารถประมาณซอฟต์แวร์ที่มีรายละเอียดมากขึ้น
- รุ่นย่อยใน COCOMO 2 ได้แก่
  - Application composition model ใช้เมื่อซอฟต์แวร์ประกอบขึ้นจาก component ที่มีอยู่
  - Early design model ใช้เมื่อมี requirement แต่การออกแบบยังไม่เริ่มต้น
  - Reuse model ใช้เพื่อคำนวณความพยายามในการรวม component ที่ใช้ซ้ำได้
  - Post-architecture model ใช้เมื่อสถาปัตยกรรมระบบได้รับการออกแบบและมีข้อมูลเพิ่มเติมเกี่ยวกับระบบ

Use of COCOMO 2 models



Application composition model

- รองรับ prototyping projects และ projects ที่มีการ reuse
- อิงจากการประมาณค่ามาตรฐานของ developer productivity ในหน่วย application (object) points/month.
- คำนึงถึงการใช้ CASE tool
- สมการคำนวณ

$$PM = (NAP \times (1 - \%reuse/100)) / PROD$$

  - PM คือค่า effort ในหน่วย person-months,
  - NAP คือค่า number of application points
  - PROD คือค่า productivity.

Object point productivity

Developer's experience and capability	Very low	Low	Nominal	High	Very
ICASE maturity and capability	Very low	Low	Nominal	High	Very
PROD (NOP/month)	4	7	13	25	5

Early design model

- วิธีการนี้จะสามารถประมาณการได้หลังจากตกลงตาม requirements
- ใช้สมการต่อไปนี้

$$PM = A \times Size^{B} \times M$$

$$M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED;$$

  - A = 2.94 เป็นค่าเริ่มต้น
  - Size คือขนาด มีหน่วยเป็น KLOC
  - B มีค่าในช่วง 1.1 ถึง 1.24 ขึ้นอยู่กับความแปลกใหม่ของโครงการ ความยืดหยุ่นในการพัฒนา แนวทางการบริหารความเสี่ยงและวุฒิภาวะของกระบวนการ (process maturity)

Multipliers

- ตัวคูณสะท้อนถึงความสามารถของนักพัฒนา non-functional requirements ความคุ้นเคยกับแพลตฟอร์มการพัฒนา ฯลฯ
  - RCPX - product reliability and complexity;
  - RUSE - the reuse required;
  - PDIF - platform difficulty;
  - PREX - personnel experience;
  - PERS - personnel capability;
  - SCED - required schedule;
  - FCIL - the team support facilities.

The reuse model

- พิจารณาเป็นกล่องดำ (Black-box) ที่ใช้ซ้ำโดยไม่มีการเปลี่ยนแปลง และไม่มีรหัสที่ต้องดัดแปลงเพื่อรวมเข้ากับรหัสใหม่
- Reuse model มีสองรุ่น:
  - Black-box นำใช้ซ้ำโดยที่ไม่มีการแก้ไขโค้ด มีการคำนวณค่าประมาณความพยายาม (PM)
  - White-box นำมาใช้ใหม่เมื่อมีการแก้ไขโค้ด คำนวณขนาดที่เทียบเท่ากับจำนวนบรรทัดของซอร์สโค้ดใหม่ จากนั้นจะปรับการประมาณขนาดสำหรับโค้ดใหม่

Reuse model estimates 1

- ใช้กับ generated code:

$$PM = (ASLOC * AT/100)/ATPROD$$

  - ASLOC คือจำนวนบรรทัดของโค้ดที่สร้างขึ้น
  - AT คือเปอร์เซ็นต์ของรหัสที่สร้างขึ้นโดยอัตโนมัติ
  - ATPROD เป็นผลงานของวิศวกรในการรวมรหัสนี้

Reuse model estimates 2

- เมื่อต้องทำความเข้าใจ code ก่อน integrate:

$$ESLOC = ASLOC * (1-AT/100) * AAM.$$

  - ASLOC และ AT เหมือนก่อนหน้านี้
  - AAM คือ adaptation adjustment multiplier คำนวณจากค่าใช้จ่ายในการเปลี่ยนโค้ดที่ reused code, ค่าใช้จ่ายในการทำความเข้าใจวิธีการรวมโค้ดและค่าใช้จ่ายในการตัดสินใจใช้ซ้ำ

Post-architecture level

- ใช้สูตรเดียวกับรูปแบบการออกแบบในช่วงต้น แต่มีตัวคูณที่เกี่ยวข้องถึง 17 ตัว
- ขนาดรหัสโดยประมาณมีดังนี้:
  - จำนวนบรรทัดของรหัสใหม่ที่จะพัฒนา
  - ประมาณการจำนวนบรรทัดของโค้ดใหม่ที่คำนวณโดยใช้รูปแบบการนำกลับมาใช้ใหม่
  - ค่าประมาณของจำนวนบรรทัดของรหัสที่ต้องแก้ไขตามการเปลี่ยนแปลงข้อกำหนด

Key points

- ไม่มีความสัมพันธ์ที่ตรงไปตรงมาระหว่างราคาที่เรียกเก็บและต้นทุนการพัฒนา
- ปัจจัยที่ส่งผลต่อผลิตภาพ (productivity) ได้แก่ ความถนัดส่วนบุคคล ประสบการณ์โดเมน โครงการพัฒนา ขนาดโครงการ การสนับสนุนเครื่องมือและสภาพแวดล้อมการทำงาน
- ในการแข่งขัน อาจมีการกำหนดราคาซอฟต์แวร์ที่จูงใจเพื่อให้ได้ชนะการประมูลและได้สัญญา เมื่อได้สัญญาแล้วก็สามารถปรับเปลี่ยนฟังก์ชันการทำงานตามราคาที่ประมูลมาได้

Key points

- ควรใช้เทคนิคต่างๆ อย่างหลากหลายในการประมาณต้นทุน เพื่อประมาณราคาซอฟต์แวร์
- ในโมเดล COCOMO มีการนำโครงการ ผลิตภัณฑ์ บุคลากร และคุณลักษณะของฮาร์ดแวร์มาพิจารณาร่วมกัน เพื่อคาดการณ์ความพยายาม (effort) ในการสร้างซอฟต์แวร์
- โมเดลต้นทุนอัลกอริธึมสามารถใช้เป็นหลักในการวิเคราะห์ค่าประมาณการเนื่องจากช่วยให้สามารถเปรียบเทียบต้นทุนของตัวเลือกต่างๆ ได้ง่ายขึ้น
- เวลาในการดำเนินโครงการมักไม่เป็นอัตราส่วนโดยตรงกับจำนวนคนที่ทำงานในโครงการ