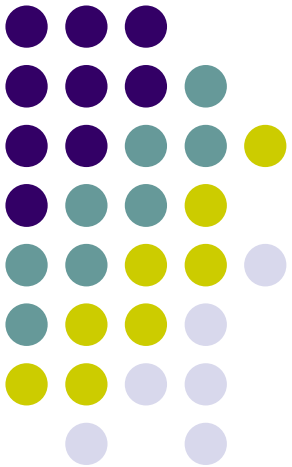
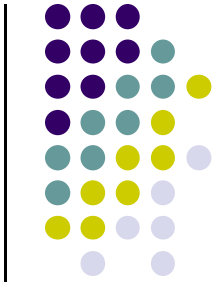
# NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

## CHƯƠNG 3 – ƯỚC LƯỢNG CHI PHÍ PHẦN MỀM

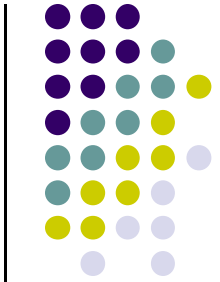


# Nội dung



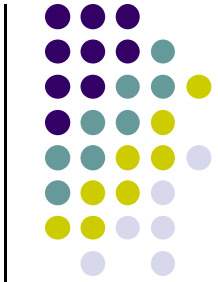
- Giới thiệu
- Ước lượng kích thước phần mềm
- Ước lượng chi phí phần mềm

# Giới thiệu



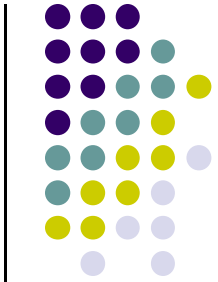
- Các yếu tố cần ước lượng
  - Kích thước phần mềm
  - Công sức phát triển
  - Thời gian thực hiện
  - Số người tham gia
- Nguyên tắc ước lượng
  - Phân rã chức năng
  - Ước lượng từng chức năng
  - Dựa trên kinh nghiệm, dữ kiện quá khứ

# Ước lượng kích thước phần mềm



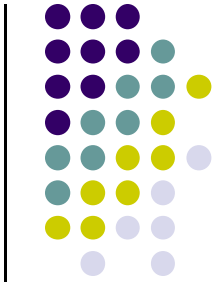
- Ước lượng kích thước phần mềm
  - Qua dòng lệnh (LOCKDSI): Ước lượng trực tiếp với từng module
  - Qua điểm chức năng (FP): Ước lượng gián tiếp thông qua ước lượng input/output, yêu cầu,…

# Ước lượng kích thước phần mềm



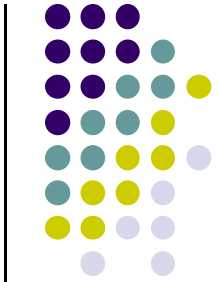
- Qua dòng lệnh
  - Theo đơn vị một dòng lệnh **LOC** (Lines Of Code)
  - Theo đơn vị một ngàn dòng lệnh **KDSI** (Thousand Delivered Source of Code)
  - Phụ thuộc ngôn ngữ lập trình

# Ước lượng kích thước phần mềm



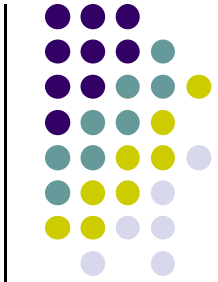
- *Các vấn đề gặp phải với các phương pháp LOC và KDSI*
  - Tính toán kích thước cho các giai đoạn khác: phân tích yêu cầu, …
  - Cài đặt trên các ngôn ngữ lập trình khác nhau: C, Java, Lisp,…
  - Cách tính số dòng mã lệnh: mã lệnh thực thi, định nghĩa dữ liệu,…
  - Sinh mã tự động, thiết kế giao diện trực tiếp (GUI)
  - *Giá thành của sản phẩm phụ thuộc vào ước lượng LOC*

# Ước lượng kích thước phần mềm



- Qua điểm chức năng (**FP** - Functional Points)
  - Độc lập với ngôn ngữ lập trình
  - Các điểm chức năng:
    - Input Item (Inp): Số input
    - Output Item (Oup): Số output
    - Inquiry (Inq): Số yêu cầu
    - Master File (Maf) : Số tập tin truy cập
    - Interface (Inf): Số giao diện

# Ước lượng kích thước phần mềm



- Bảng giá trị các điểm chức năng theo độ phức tạp từ thấp, trung bình đến cao

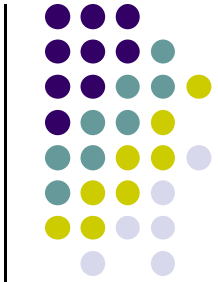
| | Level of Complexity | | |
|---|---|---|---|
| Component | Simple | Average | Complex |
| Input item *(Inp)* | 3 | 4 | 6 |
| Output item *(Out)* | 4 | 5 | 7 |
| Inquiry *(Inq)* | 3 | 4 | 6 |
| Master file *(Maf)* | 7 | 10 | 15 |
| Interface *(inf)* | 5 | 7 | 10 |

- Công thức tính số điểm chức năng thô

$$UFP = a_1 \times Inp + a_2 \times Oup + a_3 \times Inq + a_4 \times Maf + a_5 \times Inf$$

với $a_1$, $a_2$, $a_3$, $a_4$, $a_5$ là giá trị các điểm chức năng theo độ phức tạp cho trong bảng trên.

# Ước lượng kích thước phần mềm



- Ví dụ: Tính điểm chức năng thô UFP theo độ phức tạp trung bình khi thực hiện hàm tìm ước chung lớn nhất của hai số nguyên?

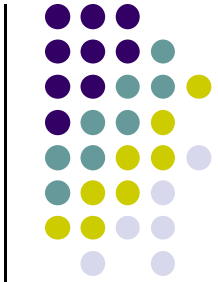
| | Level of Complexity | | |
|---|---|---|---|
| **Component** | **Simple** | **Average** | **Complex** |
| Input item *(Inp)* | 3 | 4 | 6 |
| Output item *(Out)* | 4 | 5 | 7 |
| Inquiry *(Inq)* | 3 | 4 | 6 |
| Master file *(Maf)* | 7 | 10 | 15 |
| Interface *(inf)* | 5 | 7 | 10 |

Inp = 2 Inq = 1 Inf = 0

Oup = 1 Maf = 0

UFP = 4Inp + 5Oup + 4Inq + 10Maf + 7Inf = 17

# Ước lượng kích thước phần mềm

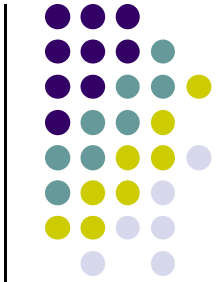


- Công thức tính điểm chức năng FP

  FP = Điểm chức năng thô x (0.65 + 0.01 x
  Tổng các mức độ ảnh hưởng của các hệ số kỹ thuật )

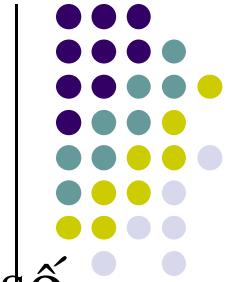
- Các hệ số kỹ thuật (có mức độ ảnh hưởng nằm trong phạm vi từ 0 (không quan trọng hay không thích hợp hay không ảnh hưởng) tới 5 (cực kỳ quan trọng hay cần thiết tuyệt đối hay ảnh hưởng nhất)
  1. Trao đổi dữ liệu (Data communication)
  2. Chức năng phân bố (Distributed function)
  3. Hiệu suất (Performance)
  4. Đặt nặng về cấu hình tiện ích (Heavily used configuration)
  5. Tỷ lệ giao tác (Transaction rate)

# Ước lượng kích thước phần mềm



- Các nhân tố kỹ thuật
  6. Trao đổi dữ liệu trực tuyến (Online data entry)
  7. Màn hình nhập liệu hiệu quả (End-user efficiency)
  8. Cập nhật trực tuyến (Online update)
  9. Xử lý phức tạp Complex processing
  10. Sử dụng lại (Reusability)
  11. Dễ cài đặt (Installation ease)
  12. Dễ thao tác (Operation ease)
  13. Được cài đặt ở nhiều tổ chức (Multiple site)
  14. Thuận lợi cho thay đổ (Facilitate change)

# Ước lượng kích thước phần mềm



- Điểm chức năng FP có thể được dùng để dự đoán số dòng lệnh LOC

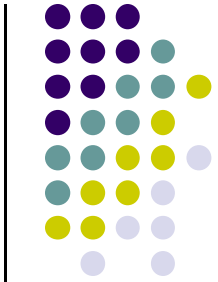
  LOC = AVC * số điểm chức năng FP

  với AVC : yếu tố phụ thuộc vào ngôn ngữ lập trình được sử dụng

| Programming Language | LOC/FP (average) |
|---|---|
| Assembly language | 320 |
| C | 128 |
| COBOL | 106 |
| FORTRAN | 106 |
| Pascal | 90 |
| C++ | 64 |
| Ada95 | 53 |
| Visual Basic | 32 |
| Smalltalk | 22 |
| Powerbuilder (code generator) | 16 |
| SQL | 12 |

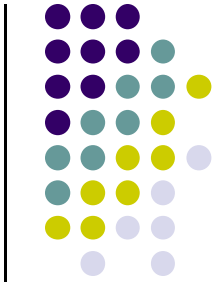
*Bảng cung cấp các dự đoán thô về LOC trung bình được yêu cầu để xây dựng một điểm chức năng ở các ngôn ngữ lập trình cấp cao*

# Ước lượng chi phí phần mềm



- Ước lượng chi phí
  - Dựa trên kích thước, độ phức tạp
  - Dựa vào dữ liệu quá khứ
  - Chi phí tỉ lệ với công sức (effort) phát triển phần mềm
  - Chi phí tính dựa theo công sức cho các giai đoạn phát triểm phần mềm: khởi đầu, phân tích, thiết kế, cài đặt, kiểm thử nhưng chưa tính đến giai đoạn bảo trì.
  - Đơn vị tính của công sức: người-tháng (hoặc người-ngày)

# Ước lượng chi phí phần mềm

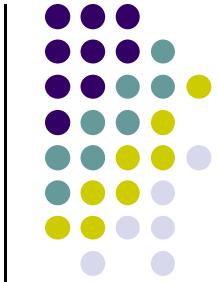


- Các phương pháp ước lượng chi phí phần mềm
  - Theo ý kiến của chuyên gia
  - Theo giải thuật
  - Bằng sự tương tự
  - Bằng luật Parkinson
  - Pricing to win

*Tự đọc*

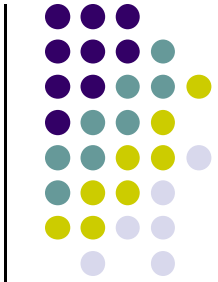
- Thực hiện các phương pháp ước lượng trên theo cách từ trên xuống hoặc từ dưới lên

# Ước lượng chi phí phần mềm



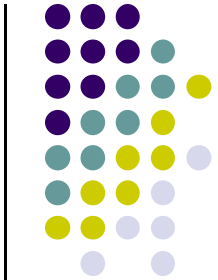
- Ước lượng theo từ trên xuống ( top-down  estimation)
  - Từ trên xuống (top — down): Khởi đầu tại mức hệ thống và đánh giá toàn thể chức năng hệ thống và cách thức chức năng này được phân phối thông qua các hệ thống con.
  - Có thể sử dụng mà không có kiến thức về kiến trúc hệ thống và các thành phần của hệ thống
  - Cần phải tính tới các chi phí như tích hợp, quản lý cấu hình và tài liệu
  - Có thể đánh giá không đúng mức chi phí giải quyết các vấn đề kỹ thuật mức thấp

# Ước lượng chi phí phần mềm



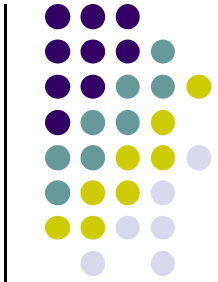
- Ước lượng theo từ dưới lên ( bottom - up  estimation)
  - Từ dưới lên (bottom - up): Khởi đầu tại mức thành phần (bộ phận của hệ thống) và ước lượng chi phí cho từng thành phần. Cộng tất cả các chi phí thành phần này để có được ước lượng cuối cùng.
  - Có thể sử dụng khi kiến trúc hệ thống được biết và các thành phần của hệ thống đã xác định
  - Phương pháp này chính xác nếu hệ thống được thiết kế một cách chi tiết
  - Có thể đánh giá không đúng mức chi phí cho các hoạt động mức hệ thống như tích hợp và tài liệu

# Ước lượng chi phí phần mềm



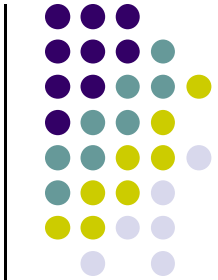
- Ý kiến của chuyên gia (expert judgment)
  - Một hay nhiều chuyên gia trong cả lĩnh vực ứng dụng và phát triển phần mềm sử dụng kinh nghiệm của họ để dự tính chi phí phần mềm. Qui trình này được lặp đi lặp lại cho đến khi đạt được sự nhất trí.
  - Thuận lợi: Phương pháp dự đoán chi phí thấp một cách tương đối. Có thể chính xác nếu các chuyên gia có kinh nghiệm trực tiếp trong các hệ thống tương tự.
  - Khó khăn: Rất thiếu chính xác nếu không có các chuyên gia thực sự!

# Ước lượng chi phí phần mềm



- Ước lượng chi phí theo giải thuật
  - Các mô hình ước lượng chi phí theo giải thuật
    - Mô hình Walston và Felix
    - Mô hình Bailey và Basili
    - Mô hình COCOMO

# Mô hình Walston và Felix



- Một trong các mô hình giải thuật sớm nhất (1977)
- Mô hình được đề nghị sau khi nghiên cứu 60 dự án của IBM
- Có 29 yếu tố ảnh hưởng tới hiệu suất
- Công thức ước lượng công sức E

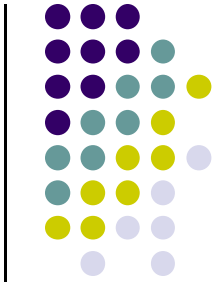
$$E = 5.25 \times S^{0.91} \text{ (người - tháng)}$$

Với S là kích thước được ước lượng của hệ thống (*theo đơn vị ngàn dòng lệnh*)

- Công thức ước lượng thời gian thực hiện

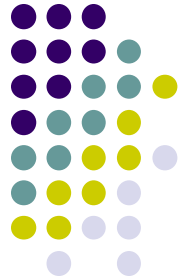
$$T = 2.5 \times E^{0.35} \text{ (tháng)}$$

# Mô hình Walston và Felix



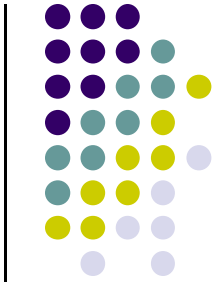
- Mỗi yếu tố sẽ nhận 1 trong 3 giá trị tùy thuộc vào sự tác động của nó tới hiệu suất
  - 1 (cao): làm tăng hiệu suất
  - 0 (trung bình): không ảnh hưởng tới hiệu suất
  - -1 (thấp): làm giảm hiệu suất

# Mô hình Walston và Felix



| | |
|---|---|
| 1. Customer interface complexity | 16. Use of design and code inspections |
| 2. User participation in requirements definition | 17. Use of top-down development |
| 3. Customer-originated program design changes | 18. Use of a chief programmer team |
| 4. Customer experience with the application area | 19. Overall complexity of code |
| 5. Overall personnel experience | 20. Complexity of application processing |
| 6. Percentage of development programmers who participated in the design of functional specifications | 21. Complexity of program flow |
| 7. Previous experience with the operational computer | 22. Overall constraints on program's design |
| 8. Previous experience with the programming language | 23. Design constraints on the program's main storage |
| 9. Previous experience with applications of similar size and complexity | 24. Design constraints on the program's timing |
| 10. Ratio of average staff size to project duration (people per month) | 25. Code for real-time or interactive operation or for execution under severe time constraints |
| 11. Hardware under concurrent development | 26. Percentage of code for delivery |
| 12. Access to development computer open under special request | 27. Code classified as nonmathematical application and input/output formatting programs |
| 13. Access to development computer closed | 28. Number of classes of items in the database per 1000 lines of code |
| 14. Classified security environment for computer and at least 25% of programs and data | 29. Number of pages of delivered documentation per 1000 lines of code |
| 15. Use of structured programming | |

# Mô hình Bailey và Felix

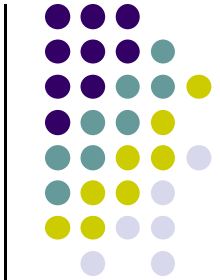


- Mô hình được đề nghị năm 1981 bởi Bailey và Felix
- Mô hình này sử dụng cơ sở dữ liệu của 18 dự án viết bằng ngôn ngữ Fortran tại trung tâm Goddard Space Flight của NASA
- Các nhóm yếu tố ảnh hưởng tới công sức: phương pháp, độ phức tạp và kinh nghiệm
- Công thức ước lượng công sức ban đầu E

$$E = 5.5 + 0.73 \times S^{1.16} \text{ (người - tháng)}$$

Với S là kích thước được ước lượng của hệ thống

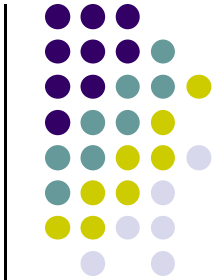
# Mô hình Bailey và Felix



- Mỗi yếu tố ảnh hưởng tới công sức nhận một trong các giá trị từ 0 đến 5

| *Total methodology (METH)* | *Cumulative complexity (CPLX)* | *Cumulative experience (EXP)* |
|---|---|---|
| Tree charts | Customer interface complexity | Programmer qualifications |
| Top-down design | Application complexity | Programmer machine experience |
| Formal documentation | Program flow complexity | Programmer language experience |
| Chief programmer teams | Internal communication complexity | Programmer application experience |
| Formal training | Database complexity | Team experience |
| Formal test plans | External communication complexity | |
| Design formalisms | Customer-initiated program design changes | |
| Code reading | | |
| Unit development folders | | |

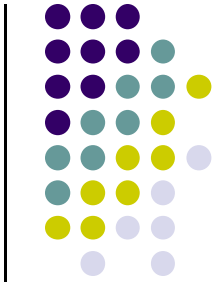
# Mô hình COCOMO 81



- Mô hình COCOMO 81 được đề nghị bởi Boehm
  - Dạng cơ bản: áp dụng cho nhóm nhỏ, môi trường quen thuộc
  - Dạng trung bình: áp dụng cho dự án khá lớn, có một ít kinh nghiệm
  - Dạng lớn: áp dụng cho dự án lớn, môi trường mới
- Bảng mức độ khó khi phát triển sản phẩm

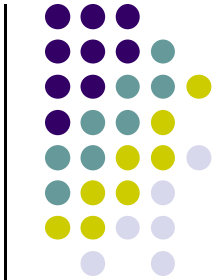
| Software project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Dạng cơ bản (organic) | 3.2 | 1.05 | 2.50 | 0.38 |
| Dạng trung bình (semi-detached) | 3.0 | 1.12 | 2.50 | 0.35 |
| Dạng lớn (embedded) | 2.8 | 1.20 | 2.50 | 0.32 |

# Mô hình COCOMO 81



- Công sức $E = a_b \times S^{b_b} \times EAF$
  - $a_b$ và $b_b$: được xác định dựa vào bảng mức độ khó khi phát triển phần mềm
  - EAF (effort adjustment factor): hệ số hiệu chỉnh công sức. Nó được tính bằng tích của các hệ số phát triển
  - S là kích thước được ước lượng của hệ thống (*theo đơn vị ngàn dòng lệnh*)
- Thời gian $T = c_b \times E^{d_b}$

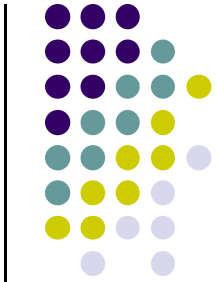
# Mô hình COCOMO 81



- Các hệ số phát triển

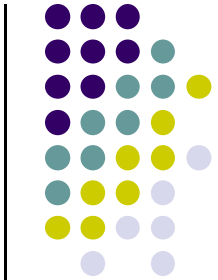
| Cost Drivers | Rating | | | | | |
|---|---|---|---|---|---|---|
| | Very Low | Low | Nominal | High | Very High | Extra High |
| **Product Attributes** | | | | | | |
| Required software reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | |
| Database size | | 0.94 | 1.00 | 1.08 | 1.16 | |
| Product complexity | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| **Computer Attributes** | | | | | | |
| Execution time constraint | | | 1.00 | 1.11 | 1.30 | 1.66 |
| Main storage constraint | | | 1.00 | 1.06 | 1.21 | 1.56 |
| Virtual machine volatility* | | 0.87 | 1.00 | 1.15 | 1.30 | |
| Computer turnaround time | | 0.87 | 1.00 | 1.07 | 1.15 | |
| **Personnel Attributes** | | | | | | |
| Analyst capabilities | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | |
| Applications experiences | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | |
| Programmer capabilitiy | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | |
| Virtual machine experience* | 1.21 | 1.10 | 1.00 | 0.90 | | |
| Programming language experiences | 1.14 | 1.07 | 1.00 | 0.95 | | |
| **Project Attributes** | | | | | | |
| Use of modern programming pratices | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | |
| Use of software tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | |
| Required development schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | |

# Mô hình COCOMO 2



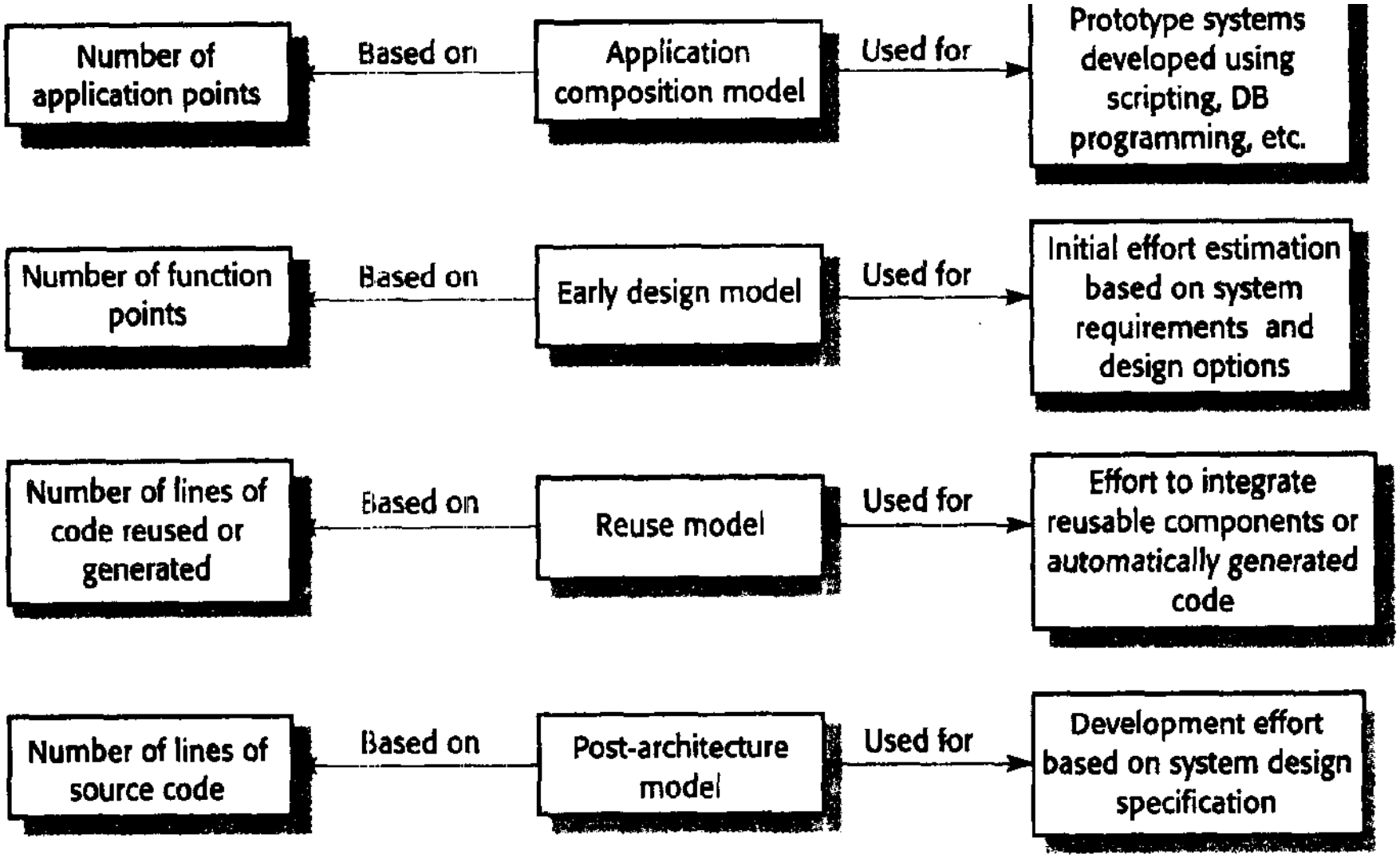
- Mô hình COCOMO 81 được phát triển trên giả thiết rằng tiến trình phát triển phần mềm thác nước được sử dụng và tất cả các phần mềm được phát triển từ đầu.
- Mô hình COCOMO 2 được thiết kế để thích ứng với những cách tiếp cận khác nhau đối với sự phát triển phần mềm

# Mô hình COCOMO 2

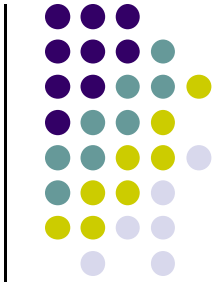


- COCOMO 2 kết hợp chặt chẽ các mô hình con nhằm đưa ra các dự đoán phần mềm chi tiết
- Các mô hình con trong COCOMO 2 là:
  - Mô hình Application composition: được sử dụng khi phần mềm được tạo thành từ các thành phần hiện có
  - Mô hình Early design: được sử dụng khi các yêu cầu là sẵn có nhưng thiết kế vẫn chưa được bắt đầu
  - Mô hình Reuse: được sử dụng để tính công sức tích hợp các thành phần có thể dùng lại được
  - Mô hình Post-architecture: được sử dụng ngay khi kiến trúc hệ thống đã được thiết kế và các thông tin chi tiết hơn về hệ thống là sẵn có

# Mô hình COCOMO 2



# Mô hình Application composition

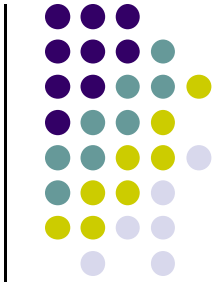


- Hỗ trợ các dự án bản mẫu và các dự án có sự tái sử dụng nhiều
- Được dựa trên số lượng điểm ứng dụng (đối tượng)
- Công thức ước lượng
  - Công sức

  E = ( NAP x (1 - %reuse/100 ) ) / PROD (người — tháng)

    - NAP: số lượng điểm ứng dụng
    - PROD: hiệu suất. Nó phụ thuộc vào kinh nghiệm và khả năng của nhà phát triển cũng như tính trưởng thành và khả năng của công cụ

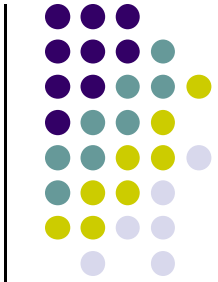
# Mô hình Application composition



- Bảng xác định hiệu suất

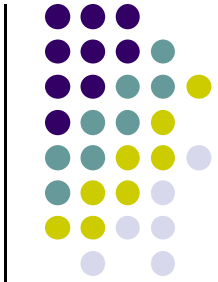
| | | | | | |
|---|---|---|---|---|---|
| Developer's experience and capability | Very low | L w | Nominal | High | Very high |
| CASE maturity and capability | Very low | Low | Nominal | High | Very high |
| PROD (NOP/month) | 4 | 7 | 13 | 25 | 50 |

# Mô hình Application composition



- Số lượng điểm ứng dụng phụ thuộc vào:
  - Các màn hình riêng biệt được hiển thị
  - Các báo cáo được sinh ra bởi hệ thống
  - Các module chương trình (được viết bằng các ngôn ngữ lập trình như Java, C++, v.v) phải được phát triển để bổ sung cho mã lệnh lập trình cơ sở dữ liệu

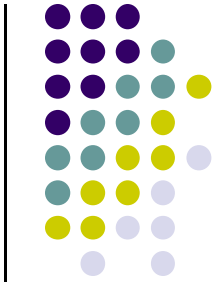
# Mô hình Application composition



- Tính số lượng điểm ứng dụng
  - Đếm số lượng màn hình, báo cáo và module
  - Xác định độ phức tạp cho từng thành phần (1 màn hình hay 1 báo cáo hay 1 module) theo bảng sau

| *For Screens* | | | | *For Reports* | | | |
|---|---|---|---|---|---|---|---|
| | *Number and source of data tables* | | | | *Number and source of data tables* | | |
| *Number of views contained* | Total < 4 (<2 server, <3 client) | Total < 8 (2-3 server, 3-5 client) | Total 8+ (>3 server, >5 client) | *Number of sections contained* | Total < 4 (<2 server, <3 client) | Total < 8 (2-3 server, 3-5 client) | Total 8+ (>3 server, >5 client) |
| <3 | simple | simple | medium | 0 or 1 | simple | simple | medium |
| 3 - 7 | simple | medium | difficult | 2 or 3 | simple | medium | difficult |
| 8 + | medium | difficult | difficult | 4 + | medium | difficult | difficult |

# Mô hình Application composition

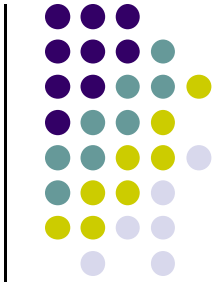


- Tính số điểm ứng dụng cho từng thành phần khi đã biết độ phức tạp theo bảng dưới đây

| *Object type* | *Simple* | *Medium* | *Difficult* |
|---|---|---|---|
| Screen | 1 | 2 | 3 |
| Report | 2 | 5 | 8 |
| 3GL component | - | - | 10 |

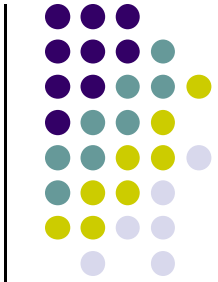
- Tính tổng số điểm ứng dụng cho tất cả các thành phần

# Mô hình Early design



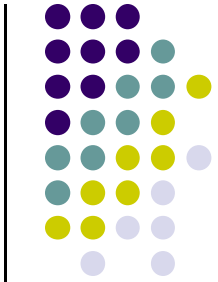
- Các ước lượng có thể được tạo ra sau khi các yêu cầu được chấp nhận
- Công thức ước lượng
  - Công sức $E = a \times S^b \times M$ với
    - M: tích của 7 hệ số nhân
    - a = 2.94
    - S kích thước được ước lượng của hệ thống (*theo đơn vị ngàn dòng lệnh*)
    - b thay đổi trong khoảng 1.1 tới 1.24 tùy thuộc vào tính mới của hệ thống, tính linh động trong phát triển, các phương pháp quản lý rủi ro và tính trưởng thành của tiến trình

# Mô hình Early design



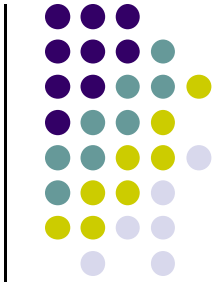
- Các hệ số nhân phản ánh khả năng của nhà phát triển, các yêu cầu phi chức năng, sự hiểu biết rõ về nền tảng phát triển, v.v.
  - RCPX - product reliability and complexity
  - RUSE - the reuse required
  - PDIF - platform difficulty
  - PREX - personnel experience
  - PERS - personnel capability
  - SCED - required schedule
  - FCIL - the team support facilities
- Giá trị của các hệ số này nằm trong khoảng từ 1 (rất thấp) đến 6 (rất cao)

# Mô hình reuse



- Đưa vào mã lệnh hộp đen được tái sử dụng mà không cần thay đổi và mã cần phải được sửa lại để tích hợp nó vào mã lệnh mới
- Hai loại tái sử dụng:
  - Tái sử dụng hộp đen: mã lệnh không được sửa đổi. Công sức phát triển cho mã hộp đen được tính là 0
  - Tái sử dụng hộp trắng: mã lệnh được chỉnh sửa để nó có thể hoạt động trong hệ thống mới một cách chính xác. Một số công sức phát triển được cần đến.

# Mô hình reuse

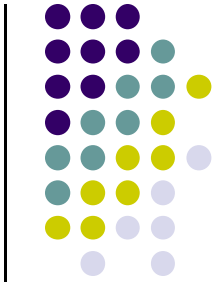


- Nhiều hệ thống còn bao gồm các mã lệnh được sinh ra một cách tự động từ các bộ dịch chương trình (một dạng tái sử dụng)
- Dự đoán công sức để tích hợp mã lệnh được sinh ra tự động:

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

  - ASLOC: số dòng mã lệnh trong các thành phần mà chúng được sửa lại cho phù hợp
  - AT: tỷ lệ phần trăm của mã lệnh được sinh tự động
  - ATPROD: hiệu suất của các kỹ sư trong việc tích hợp mã lệnh này

# Mô hình Reuse

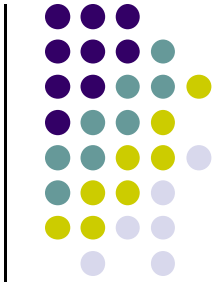


- Dự đoán công sức để tích hợp các mã lệnh mới và các thành phần tái sử dụng hộp trắng:
  - Không dự đoán công sức trực tiếp mà qua số dòng mã nguồn mới

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

    - ESLOC: số dòng mã nguồn mới tương đương
    - ASLOC và AT như trước
    - AAM: hệ số nhân hiệu chỉnh sự thích ứng từ các chi phí của việc thay đổi mã lệnh tái sử dụng, các chi phí để hiểu cách tích hợp mã lệnh và các chi phí để ra quyết định tái sử dụng

# Mô hình Post-architecture



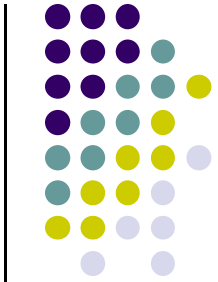
- Sử dụng cùng một công thức như mô hình early design nhưng có tới 17 hệ số nhân
  - Công sức $E = a \times S^b \times M$ với
    - M: tích của 17 hệ số nhân
    - a = 2.94
    - S kích thước được ước lượng của hệ thống (*theo đơn vị ngàn dòng lệnh*)
    - $b = 1.01 + 0.01 \times \sum W_i$ với $W_i$ là hệ số có giá trị biến đổi từ 5 (rất thấp) tới 0 (cực kỳ cao)

# Mô hình Post-architecture

- 17 hệ số nhân M

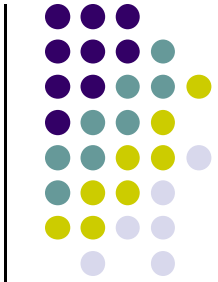
| Cost Drivers | Rating | | | | | |
|---|---|---|---|---|---|---|
| | Very Low | Low | Nominal | High | Very High | Extra High |
| Product factors | | | | | | |
| Reliability required | 0.75 | 0.88 | 1.00 | 1.15 | 1.39 | |
| Database size | | 0.93 | 1.00 | 1.09 | 1.19 | |
| Product complexity | 0.70 | 0.88 | 1.00 | 1.15 | 1.30 | 1.66 |
| Required reusability | | 0.91 | 1.00 | 1.14 | 1.29 | 1.49 |
| document needs | 0.89 | 0.95 | 1.00 | 1.06 | 1.13 | |
| Computer factors | | | | | | |
| Execution time constraint | | | 1.00 | 1.11 | 1.31 | 1.67 |
| Main storage constraint | | | 1.00 | 1.06 | 1.21 | 1.57 |
| Platform volatility | | 0.87 | 1.00 | 1.15 | 1.30 | |
| Personnel factors | | | | | | |
| Analyst capabilities | 1.50 | 1.22 | 1.00 | 0.83 | 0.67 | |
| Programmer capability | 1.37 | 1.16 | 1.00 | 0.87 | 0.74 | |
| Applications experiences | 1.22 | 1.10 | 1.00 | 0.89 | 0.81 | |
| Platform experience | 1.24 | 1.10 | 1.00 | 0.92 | 0.84 | |
| Language and tool experiences | 1.25 | 1.12 | 1.00 | 0.88 | 0.81 | |
| Personal continuity | 1.24 | 1.10 | 1.00 | 0.92 | 0.84 | |
| Project factors | | | | | | |
| Use of software tools | 1.24 | 1.12 | 1.00 | 0.86 | 0.72 | |
| Multi-site development | 1.25 | 1.10 | 1.00 | 0.92 | 0.84 | 0.78 |
| Required development schedule | 1.29 | 1.10 | 1.00 | 1.00 | 1.00 | |

# Mô hình Post-architecture



- Kích thước mã lệnh S trong mô hình này được tính bằng cách cộng ba ước lượng dưới đây:
  - Sự ước lượng về số dòng mã nguồn mới được phát triển
  - Sự ước lượng về số dòng mã nguồn tương đương được tính bằng cách sử dụng mô hình reuse
  - Sự ước lượng về số dòng mã lệnh phải được sửa đổi theo các thay đổi về yêu cầu

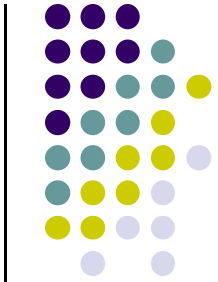
# Mô hình Post-architecture



- Các hệ số W

| | |
|---|---|
| Precedentedness | Reflects the previous experience of the organisation with this type of project. Very low means no previous experience; Extra high means that the organisation is completely familiar with this application domain. |
| Development flexibility | Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client sets only general goals. |
| Architecture/risk resolution | Reflects the extent of risk analysis carried out. Very low means little analysis; Extra high means a complete and thorough risk analysis. |
| Team cohesion | Reflects how well the development team know each other and work together. Very low means very difficult interactions; Extra high means an integrated and effective team with no communication problems. |
| Process maturity | Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5. |

# Mô hình Post-architecture



- Các hệ số W

| W(i) | Very Low | Low | Nominal | High | Very High | Extra High |
|---|---|---|---|---|---|---|
| Precedentedness | 4.05 | 3.24 | 2.43 | 1.62 | 0.81 | 0.00 |
| Development Flexibility | 6.07 | 4.86 | 3.64 | 2.43 | 1.21 | 0.00 |
| Architecture / Risk Resolution | 4.22 | 3.38 | 2.53 | 1.69 | 0.84 | 0.00 |
| Team Cohesion | 4.94 | 3.95 | 2.97 | 1.98 | 0.99 | 0.00 |
| Process Maturity | 4.54 | 3.64 | 2.73 | 1.82 | 0.91 | 0.00 |