Component-based software engineering

Week 13

ดัดแปลงจาก slides ของหนังสือ Software Engineering [1]

หัวข้อที่จะศึกษา

- Components and component models
- The CBSE process
- Component composition

Component-based development

- วิศวกรรมซอฟต์แวร์แบบอิงส่วนประกอบ (CBSE) เป็นแนวทางในการพัฒนาซอฟต์แวร์ที่ต้องอาศัยการนำซอฟต์แวร์มาใช้ซ้ำ
- การ reuse โดยใช้ object เดี่ยว (หรือ class) ในการพัฒนาเชิงวัตถุ ไม่สามารถรองรับการนำกลับมาใช้ใหม่อย่างมีประสิทธิภาพ
  - คลาสเดี่ยวมักมีรายละเอียดและเฉพาะเจาะจงเกินไปและไม่ครอบคลุมการใช้งาน
- Component มีความเป็นนามธรรมมากกว่า class object
  - สามารถใช้งานได้แบบ stand alone (อาจจะไม่ต้องพึ่ง component อื่น ๆ)

สิ่งสำคัญใน CBSE

- Independent components
  - ระบุความแตกต่างได้จาก interfaces
- Component standards
  - เพื่อให้สามารถใช้งาน component หลายตัวร่วมกันได้
- Middleware
  - ที่ช่วยให้ component ทำงานร่วมกัน
- A development process
  - ที่รองรับและมุ่งไปที่การ reuse

CBSE และหลักการออกแบบ

- นอกจากประโยชน์ของการนำกลับมาใช้ใหม่แล้ว CBSE ยังต้องยึดหลักการออกแบบทางวิศวกรรมซอฟต์แวร์ที่ดี:
  - Component มีความเป็นอิสระและไม่รบกวนซึ่งกันและกัน
  - มีการซ่อนรายละเอียดใน component
  - มีการกำหนดรูปแบบการสื่อสารผ่าน Interface ของ component ไว้เป็นอย่างดี
  - มีการใช้แพลตฟอร์มร่วมกันเพื่อลดต้นทุนในการพัฒนา

ปัญหาของ CBSE

- ความน่าเชื่อถือของ component
  - ส่วนประกอบที่ไม่มีซอร์สโค้ดมาให้ จะมีความน่าเชื่อถือได้แค่ไหน
- การรับรอง component
  - ใครจะเป็นผู้รับรองคุณภาพของ component
- การคาดเดาคุณสมบัติฉุกเฉิน
  - หากมีเหตุการณ์ภายนอกที่ส่งผลกระทบต่อ component ซึ่งไม่ได้รับการออกแบบไว้ล่วงหน้า จะมีการตอบสนองจาก component อย่างไร
- ข้อเปรียบเทียบระหว่าง component
  - ถ้ามี component ชนิดเดียวกันให้เลือกใช้จากหลายแหล่ง เราจะมีวิธีการเลือกใช้อย่างเหมาะสมได้อย่างไร

Components คืออะไร

- Component คือชิ้นส่วน (entity) หนึ่งของ software ที่ให้บริการในระบบ
  - ไม่คำนึงถึงตำแหน่งที่ตั้งของ component หรือภาษาโปรแกรมที่ใช้สร้าง component นั้น
- Component คือชิ้นส่วน ที่สามารถเรียกทำงานได้อิสระ
  - สามารถประกอบขึ้นจาก object ที่สามารถเรียกใช้งานได้ตั้งแต่หนึ่งตัวขึ้นไปทำงานร่วมกัน
- เมื่อติดตั้ง component ในระบบแล้ว อินเทอร์เฟซของคอมโพเนนต์จะได้รับการเผยแพร่ (published)
  - การโต้ตอบ (interaction) ทั้งหมดจะทำผ่านอินเทอร์เฟซที่เผยแพร่เหล่านั้น

Component definitions

- Councill and Heinmann:
  - A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.
- Szyperski:
  - A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.

Component ในฐานะ service provider

- Component มีคุณสมบัติ
  - ปฏิบัติการได้อิสระ (independent)
  - ไม่จำเป็นต้องคอมไพล์ก่อนที่จะใช้กับ component อื่นๆ
- Component ให้บริการผ่าน interface
- Component ต่างๆ ทำงานร่วมกันผ่าน interface
- จะต้องไม่มี component ใดที่เรียกใช้ส่วนที่ไม่ได้ published ของ component อื่น ๆ

คุณสมบัติที่สำคัญของ Component (1)

- สามารถประกอบ component เข้าด้วยกันได้ (Composable)
  - การโต้ตอบภายนอกทั้งหมดต้องเกิดขึ้นผ่าน interface ที่กำหนดไว้แบบสาธารณะ (public)
  - component ต้องจัดเตรียมวิธีการเข้าถึงข้อมูลเกี่ยวกับตัวเองจากภายนอก เช่น methods และ properties
- ปรับใช้ได้ (Deployable)
  - Component ต้องมีส่วนประกอบที่ใช้งานได้อยู่ในตัวเอง
  - เป็น entity แบบ stand alone บน platform ที่นำ component ไปใช้งาน
  - มักอยู่ในรูป binary ของภาษาเครื่องคอมพิวเตอร์ ซึ่งมักจะหมายความว่า component นั้นสามารถเรียกใช้ได้ทันที ไม่ต้องคอมไพล์ก่อนนำไปใช้งาน
  - หากจำเป็นต้องมีการปรับปรุง component จะต้องกระทำโดยเจ้าของ component ไม่ใช่ user ของ component นั้น

คุณสมบัติที่สำคัญของ Component (2)

- คู่มือการใช้ (Documents)
  - ต้องจัดทำเอกสารคู่มืออย่างสมบูรณ์เพื่อให้ผู้ใช้สามารถตัดสินใจได้ว่าตรงตามความต้องการหรือไม่
  - ควรระบุความหมายของ interface ของ component ทั้งหมด
- มีความเป็นอิสระ (Independent)
  - Component ควรเป็นอิสระ
  - สร้างและปรับใช้ (deploy) โดยไม่ต้องนำ component อื่น ๆ มาเกี่ยวข้อง
  - ในกรณีที่ component ต้องการ service จากภายนอก จะต้องมีการกำหนดไว้อย่างชัดเจนในหัวข้อ interface ที่ชื่อ requires

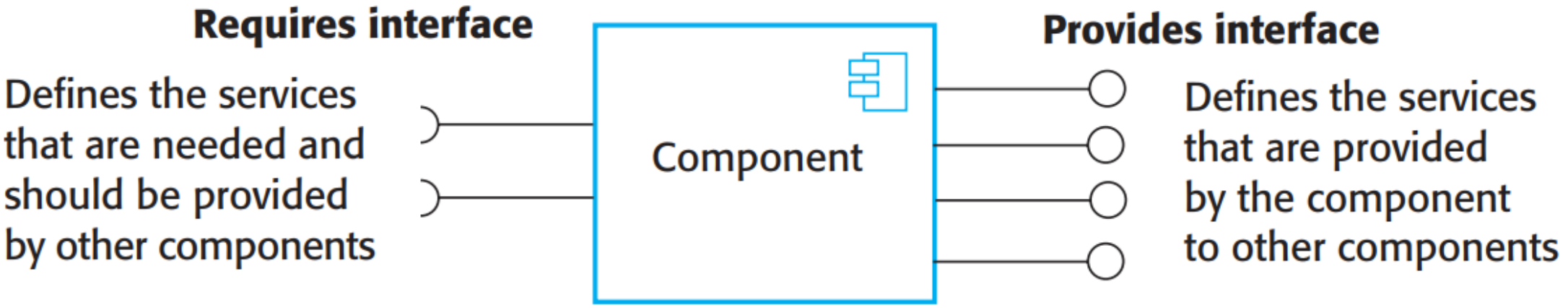
คุณสมบัติที่สำคัญของ Component (3)

- มีมาตรฐาน (Standardized)
  - Component ที่ได้มาตฐานคือ component ที่ทำตามกระบวนการ CBSE และเข้ากันได้กับ standard component model
  - standard component model จะระบุ component interfaces, component metadata, documentation, composition, และ component deployment

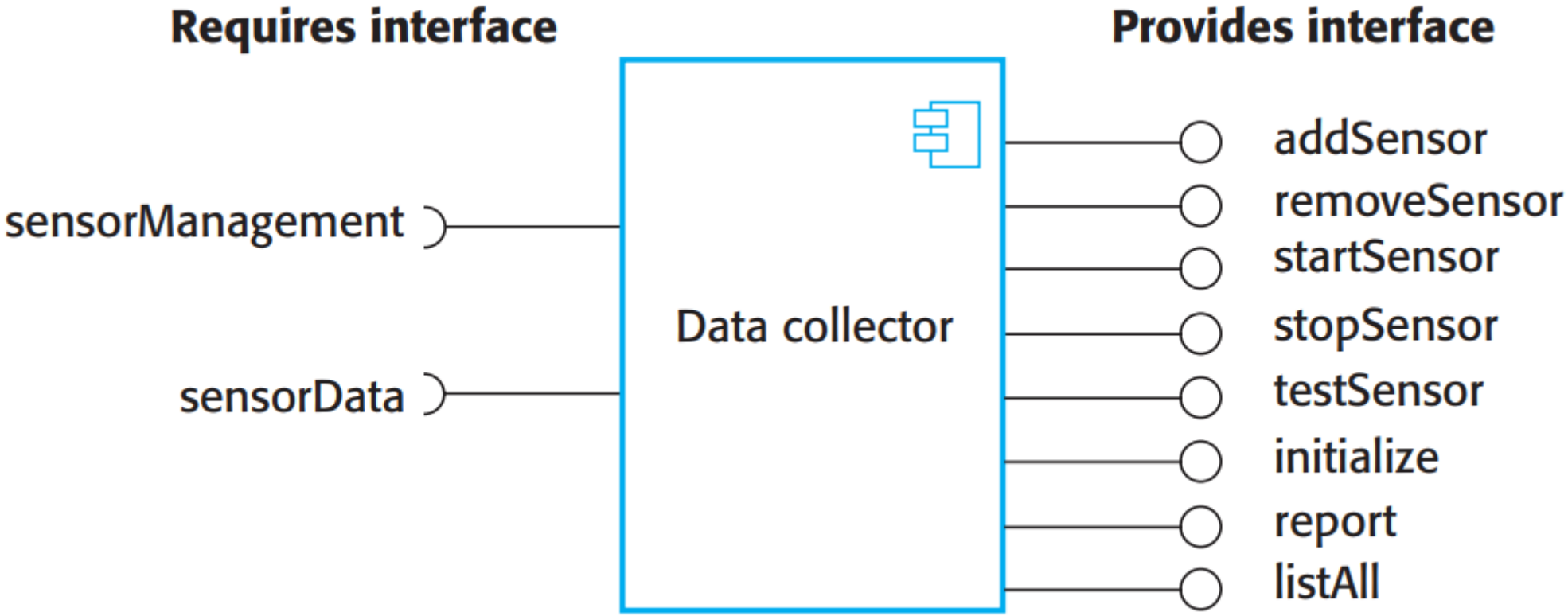
Component interfaces

- Provides interface
  - ระบุบริการ (services) ที่จัดเตรียมไว้โดย component เพื่อให้ component อื่น ๆ เรียกใช้
- Requires interface
  - ระบุบริการ (services) ที่ระบบต้องจัดเตรียมไว้ เพื่อให้ component นี้สามารถให้บริการได้

Component interfaces



ตัวอย่าง Component interfaces



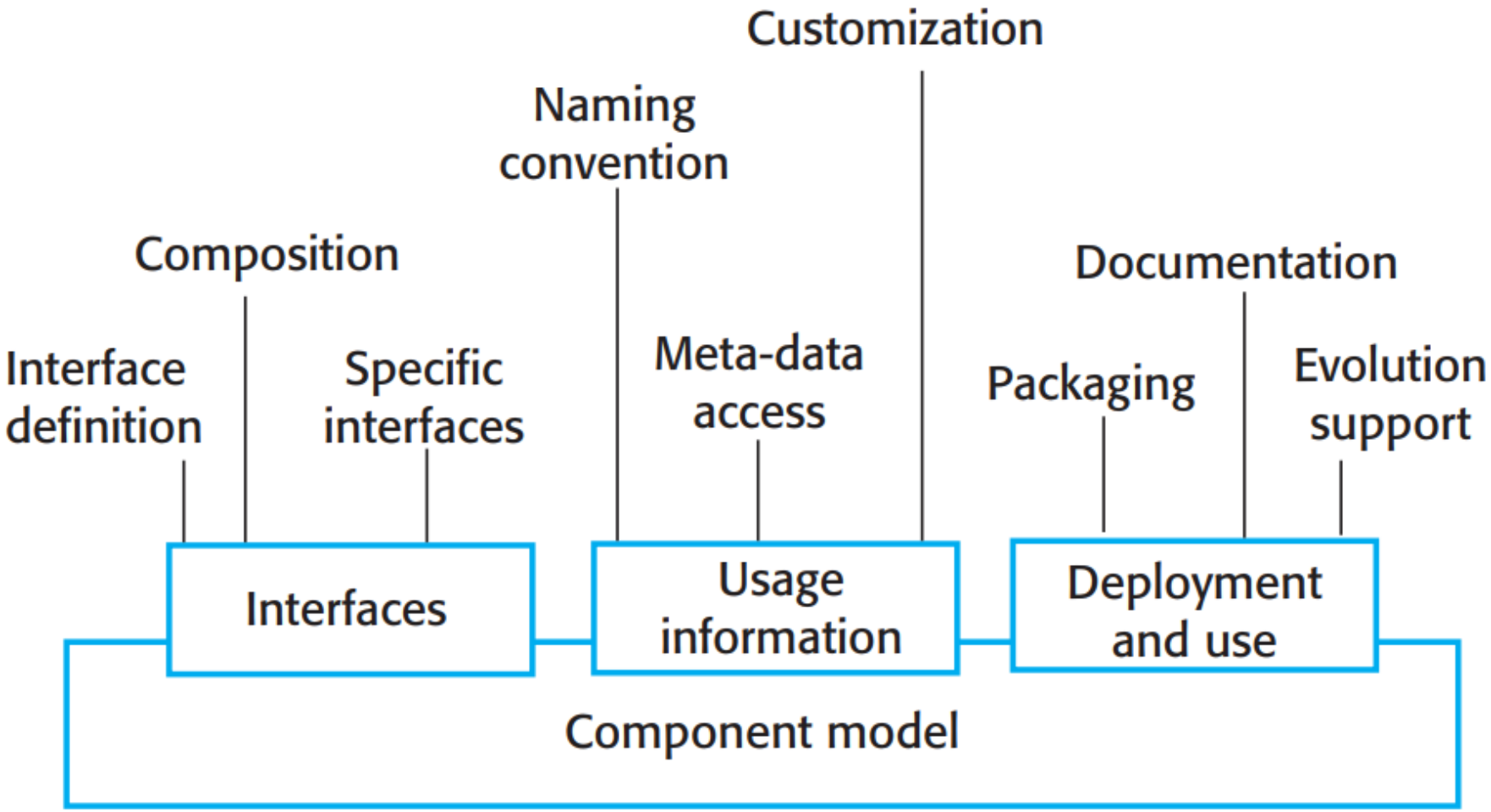
Components and objects

- Components เป็น deployable entities
- Components ไม่กำหนด types
- Component ถูกสร้างให้ปกปิดรายละเอียดภายในมากที่สุด (opaque)
- Components เป็นอิสระจากภาษาโปรแกรม (language-independent)
- Components มีมาตรฐาน (standardized)

Components and objects

- Component model คือนิยามมาตรฐานสำหรับการใช้สร้าง เอกสารประกอบ และการใช้งาน component
- ตัวอย่าง component model
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - Corba model
- Component model จะระบุวิธีกำหนด interface และ component ที่ควรรวมอยู่ใน interface นั้น

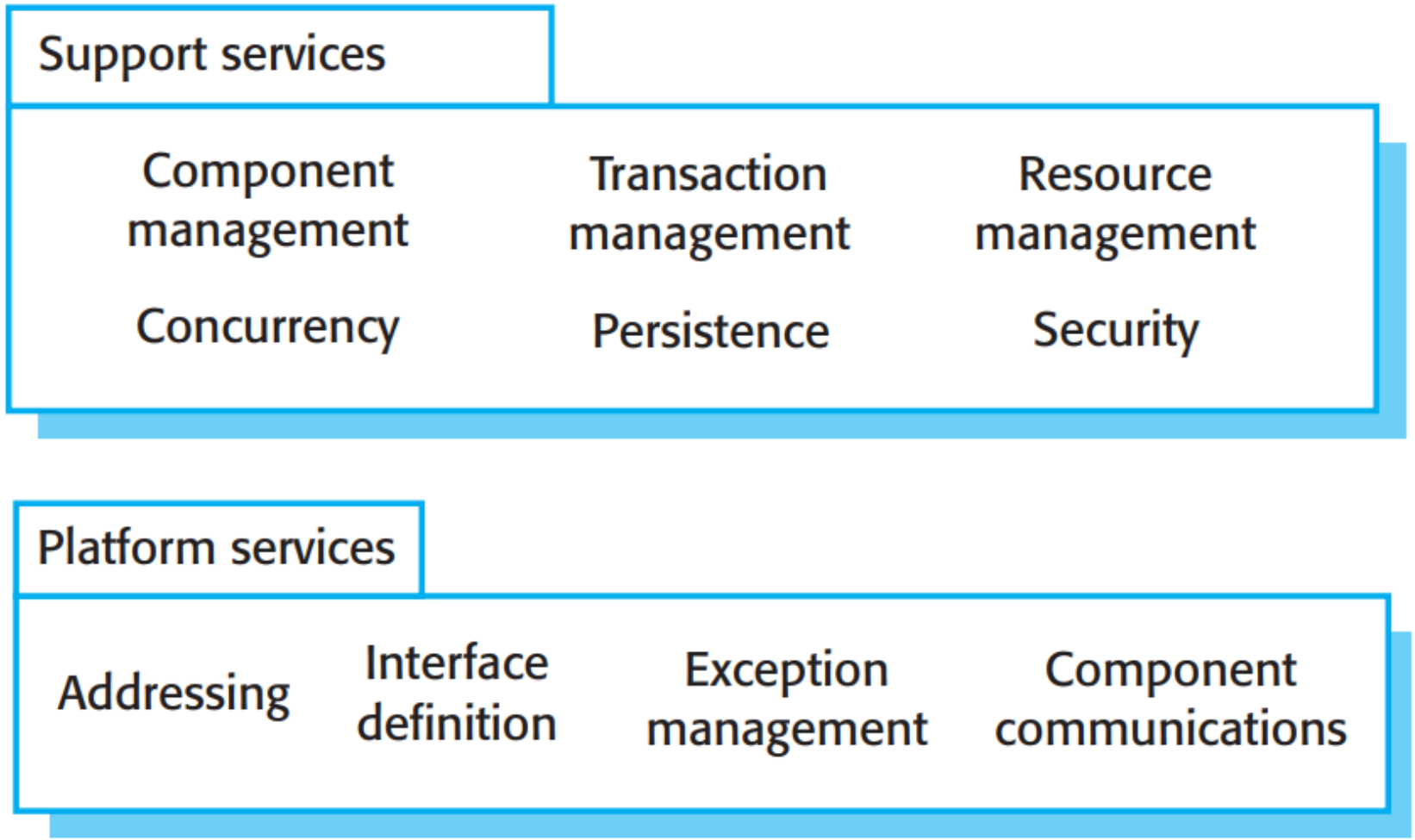
Elements of a component model



Middleware support

- Component model เกิดจากการรวมกันของ middleware พื้นฐานที่ให้การสนับสนุนสำหรับการดำเนินการแก่ component
- Component model ให้บริการต่อไปนี้
  - บริการแพลตฟอร์ม (platform services) ที่ทำให้ component สามารถสื่อสารได้
  - บริการสนับสนุน (Support services) เป็นบริการที่ไม่ขึ้นกับแอปพลิเคชัน เรียกใช้โดย component ต่างๆ
- ในการเรียกใช้ service ต่างๆ ภายใน model นั้น component จะถูกวางไว้ในคอนเทนเนอร์ (container) ซึ่งจะมีชุดของ interface ที่ใช้ในการเข้าถึงการใช้ services ของ component

Middleware support



Component development for reuse

- Component ที่พัฒนาขึ้นสำหรับแอปพลิเคชันเฉพาะ (specific application) จะต้องทำให้เป็นแบบทั่วไปเพื่อให้นำมาใช้ใหม่ได้
- Component จะใช้ซ้ำได้มากที่สุดก็ต่อเมื่อมันสามารถอธิบายเป็นนิยามที่เสถียรได้ (business object)
  - ตัวอย่างเช่น นามธรรมของโดเมนที่มีเสถียรภาพของโรงพยาบาล ได้แก่ พยาบาล ผู้ป่วย การรักษา ฯลฯ

Component development for reuse

- Components ที่สามารถ reuse อาจจะสร้างจากการ components ที่มีอยู่ ให้อยู่ในรูปทั่วไป
- Component reusability
  - ควรเป็นนิยามที่มั่นคง
  - ปกปิดการทำงานภายใน
  - มีอิสระจาก component อื่นๆ ให้มากที่สุด
  - มีการ publish การรับมือต่อความผิดพลาด (exception) ผ่านทาง component interface
- การชั่งน้ำหนักระหว่าง reusability และ usability
  - เมื่อมีความเป็น general ของ interface ที่มากขึ้น จะทำให้ reusability สูงขึ้นก็จริง แต่จะเพิ่มความซับซ้อนและอาจจะใช้งานจริงได้น้อยลง

Changes for reusability

- ลบ application-specific methods
- เปลี่ยนชื่อเพื่อทำให้มีความเป็น general มากขึ้น
- เพิ่ม methods ให้รองรับการทำงานที่มากขึ้น กว้างขึ้น
- ทำให้ exception handling มีความคงเส้นคงวา
- เพิ่ม configuration interface เพื่อการปรับเปลี่ยน component ในอนาคต
- รวมส่วนประกอบต่าง ๆ ที่จำเป็นไว้ในตัวเอง เพื่อลด dependencies กับ component ภายนอก

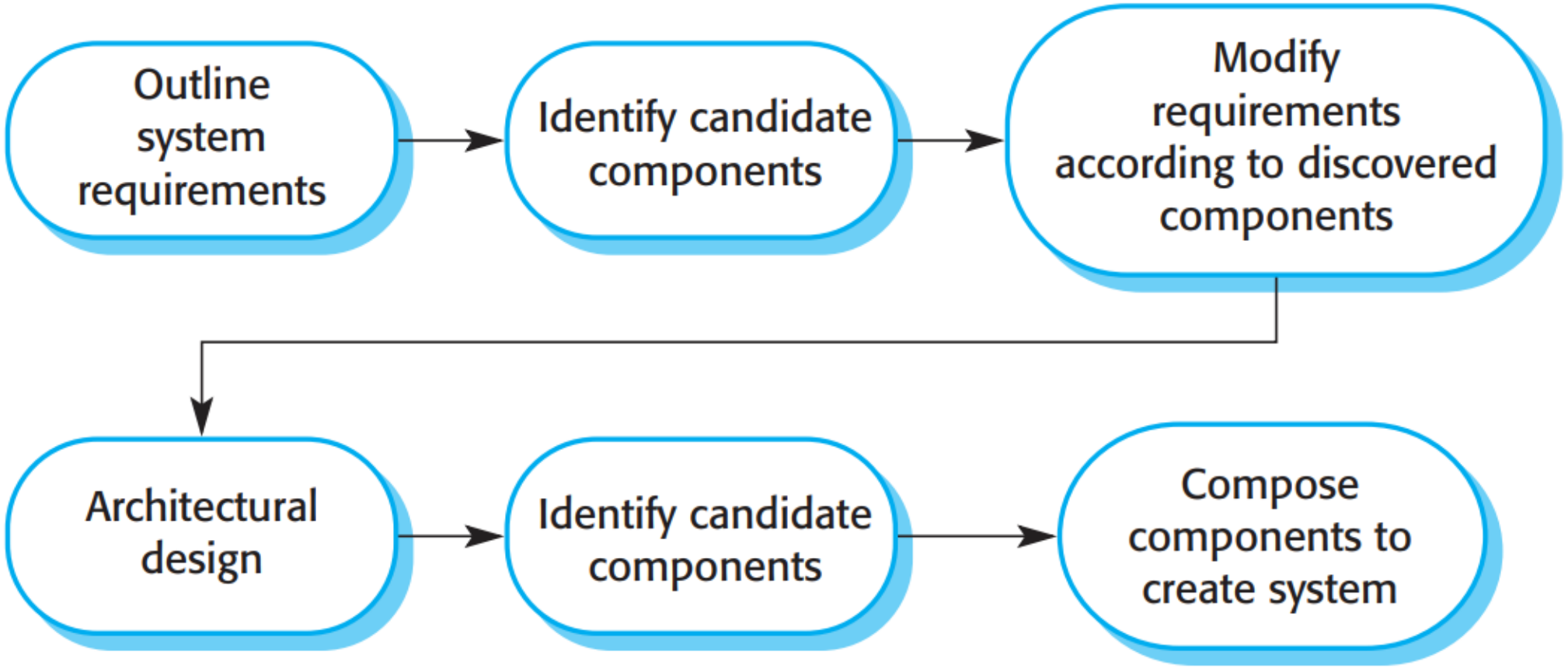
Legacy system components

- ระบบเดิม (Legacy system) ซึ่งตอบสนองฟังก์ชันทางธุรกิจ สามารถบรรจุใหม่ (re-packaged) ใน component เพื่อนำกลับมาใช้ใหม่ได้
  - เป็นงานเกี่ยวข้องกับการเขียน wrapper ให้กับ component ด้วยวิธีการเขียน interface เพื่อเข้าใช้ส่วนสำคัญของระบบเดิม
  - แม้ว่าจะมีต้นทุนสูงแต่ก็อาจคุ้มค่ากว่าการลงมือพัฒนาระบบเดิมใหม่

Reusable components

- ต้นทุนการพัฒนา component ที่สามารถ reuse ได้ อาจสูงกว่าต้นทุนของการพัฒนาแบบปกติ
- ต้นทุนการปรับปรุงความสามารถในการ reuse ที่เพิ่มเติมเข้ามานี้ ควรเป็นต้นทุนระดับองค์กรมากกว่าต้นทุนของโครงการ
- การพัฒนา component เพื่อให้ใช้งานได้แบบ generic อาจมีเวลาดำเนินการนานกว่า component ที่เทียบเท่ากัน

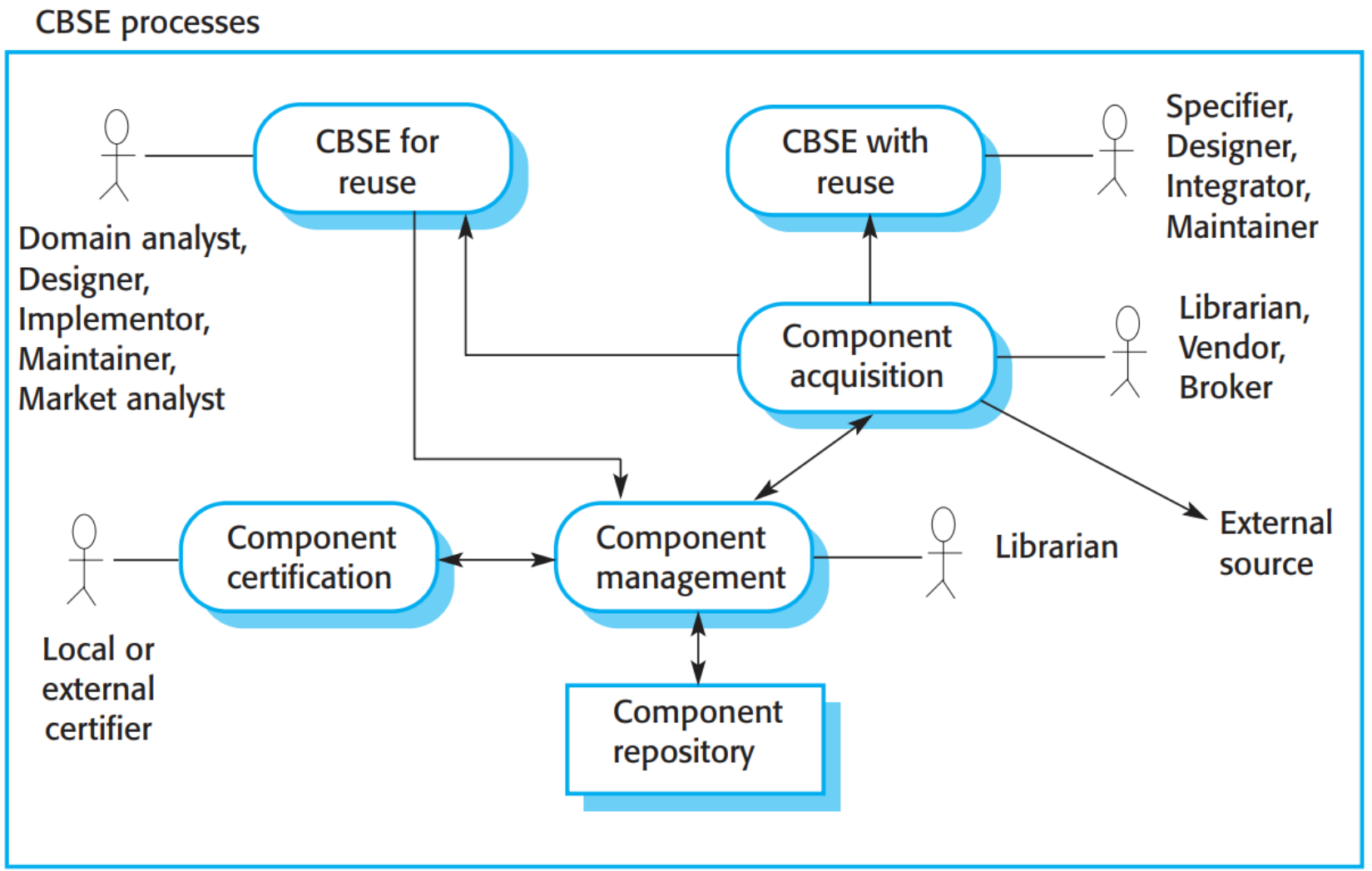
The CBSE process



The CBSE process

- เมื่อจะดำเนินการพัฒนาแบบ component reuse จำเป็นต้องทำการตัดสินใจเลือกระหว่าง ความต้องการในทางอุดมคติกับบริการที่ component ที่มีอยู่จริงสามารถให้บริการได้
- ลำดับการดำเนินการ
  1. การพัฒนาโครงร่างของ requirement
  2. ค้นหา component แล้วแก้ไข requirement ตามฟังก์ชันที่มีใน component
  3. ค้นหาอีกครั้งเพื่อดูว่ามี component ที่ดีกว่า component ที่ได้ในข้อ 2) หรือไม่
- รูปแบบการดำเนินการ
  - Development for reuse
  - Development by reuse

The CBSE process



The component identification process

1 Component acquisition

เป็นกระบวนการจัดหา component เพื่อนำมา reuse หรือพัฒนาให้เป็น component ที่ reuse ได้

2. Component management

เกี่ยวข้องกับการจัดการการ reuse เพื่อให้แน่ใจว่า component ได้รับการจัดหมวดหมู่อย่างเหมาะสม มีการจัดเก็บ และทำให้พร้อมสำหรับการ reuse

3. Component certification

เป็นกระบวนการตรวจสอบ component และการรับรองว่าเป็นไปตามข้อกำหนด

Component identification issues

- ความน่าเชื่อถือ (Trust)
  - ต้องทำให้แน่ใจว่าสามารถไว้วางใจ supplier ของ component ได้อย่างดีที่สุด
  - ส่วนประกอบที่ไม่น่าเชื่อถืออาจไม่ทำงานตามที่โฆษณาไว้ และที่เลวร้ายที่สุด มันสามารถละเมิดความปลอดภัยของเราได้
- Requirements.
  - ไม่มี component ใดที่ตอบสนองความต้องการได้ครบถ้วน
  - การเลือก component เป็นกลุ่มเพื่อมาทำงานร่วมกัน ต้องแน่ใจว่าตอบสนอง requirement ได้โดยไม่ขัดแย้งกันเอง

Component identification issues

- Validation.
  - รายละเอียดหรือข้อมูลจำเพาะของ component อาจไม่มีรายละเอียดเพียงพอที่จะทำให้เชื่อได้ว่ามีการทดสอบอย่างครอบคลุม
  - Component อาจมีการทำงานที่ไม่ต้องการติดมาด้วย เราจะทดสอบสิ่งนี้ได้อย่างไรว่าจะไม่รบกวน application ของเราในอนาคต

Ariane launcher failure

The Ariane 5 launcher failure

While developing the Ariane 5 space launcher, the designers decided to reuse the inertial reference software that had performed successfully in the Ariane 4 launcher. The inertial reference software maintains the stability of the rocket. The designers decided to reuse this without change (as you would do with components), although it included additional functionality that was not required in Ariane 5.

In the first launch of Ariane 5, the inertial navigation software failed, and the rocket could not be controlled. The rocket and its payload were destroyed. The cause of the problem was an unhandled exception when a conversion of a fixed-point number to an integer resulted in a numeric overflow. This caused the runtime system to shut down the inertial reference system, and launcher stability could not be maintained. The fault had never occurred in Ariane 4 because it had less powerful engines and the value that was converted could not be large enough for the conversion to overflow.

This illustrates an important problem with software reuse. Software may be based on assumptions about the context where the system will be used, and these assumptions may not be valid in a different situation.

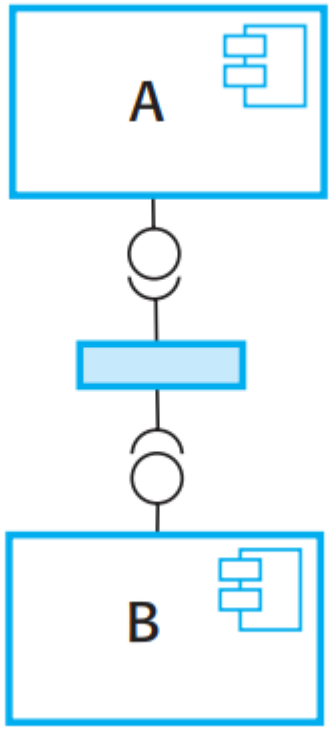
More information about this failure is available at: http://software-engineering-book.com/case-studies/ariane5/

Component composition

- เป็นขั้นตอนการประกอบชิ้นส่วนเพื่อสร้างระบบ
- งานที่ต้องดำเนินการเกี่ยวข้องกับการรวม component ต่าง ๆ เข้าด้วยกันและรวมเข้ากับโครงสร้างพื้นฐานของส่วนประกอบ
- โดยปกติเราต้องเขียน ‘glue code' เพื่อรวม component ต่าง ๆ เข้าด้วยกัน

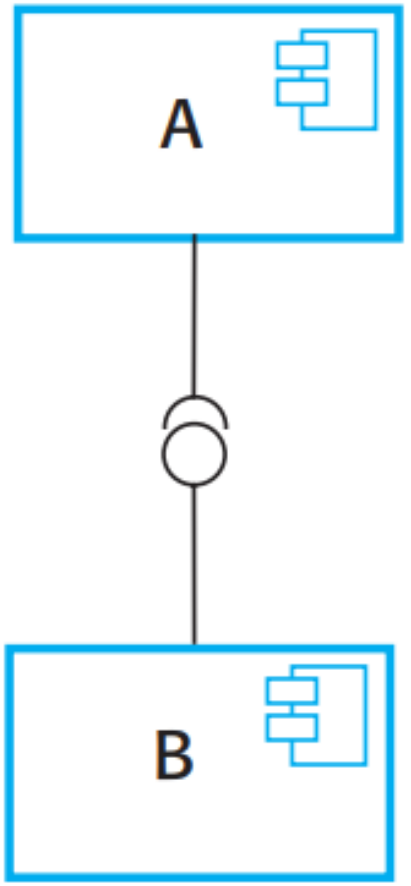
Types of composition

- Sequential composition
  - component ที่ประกอบเข้าด้วยกันจะถูกดำเนินการ (เรียกใช้งาน) ตามลำดับ ขึ้นอยู่กับการเขียน interface ที่จัดเตรียมไว้ของแต่ component



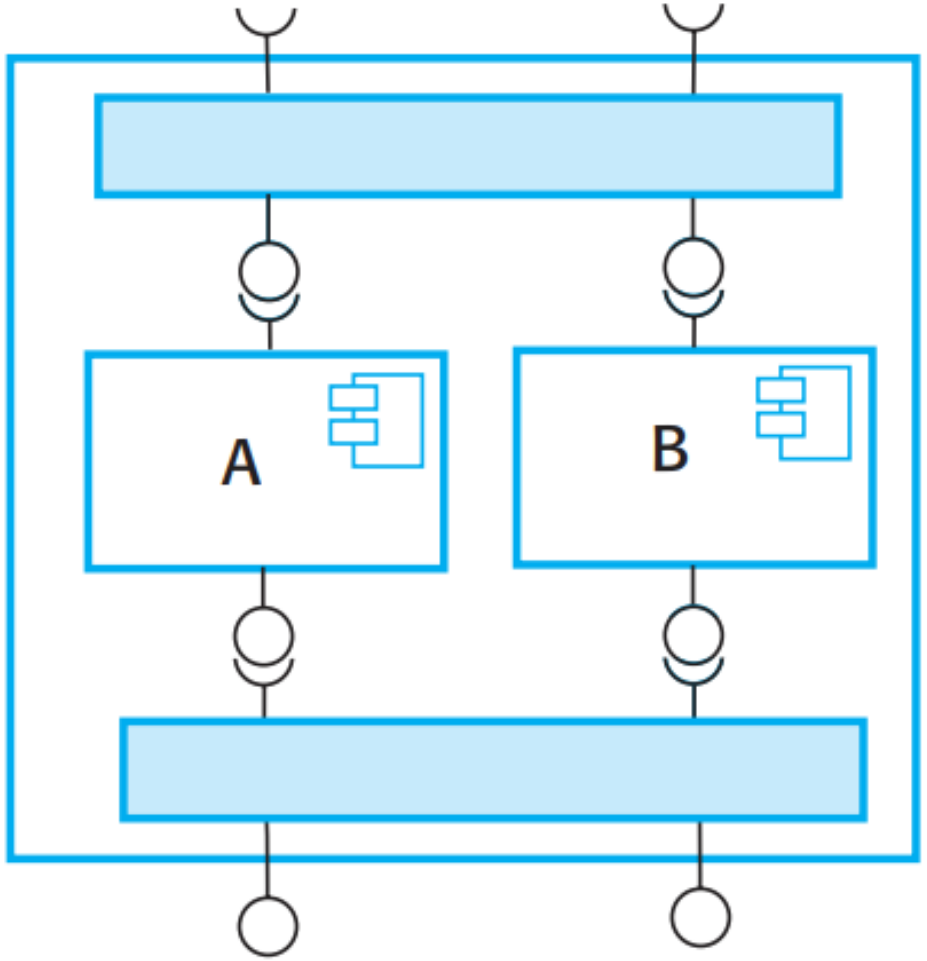
Types of composition

- Hierarchical composition
  - Component หนึ่งจะเรียกใช้บริการของอีก component หนึ่ง โดย provide interface ของ component ตัวหนึ่งจะประสานกับ requires interface ของ component อีกตัวหนึ่ง



Types of composition

- Additive composition
  - องค์ประกอบเสริมที่ interface ของสอง component ถูกรวมเข้าด้วยกันเพื่อ component ใหม่



Interface incompatibility

- Parameter incompatibility
  - การดำเนินการมีพารามิเตอร์ซึ่งมีชื่อเหมือนกันแต่ประเภทข้อมูลแตกต่างกัน
- Operation incompatibility
  - ชื่อของการดำเนินการใน composed interfaces ต่างกัน
- Operation incompleteness
  - provide interface ของ component หนึ่งเป็นส่วนย่อยของ requires interface ของ component อื่น

Adaptor components

- แก้ไขปัญหาความไม่ลงรอยกันของ interface โดยการปรับ interface ของ component ที่ประกอบขึ้นใหม่
- บางครั้งต้องใช้อะแดปเตอร์หลากหลาย ขึ้นอยู่กับประเภทของ component

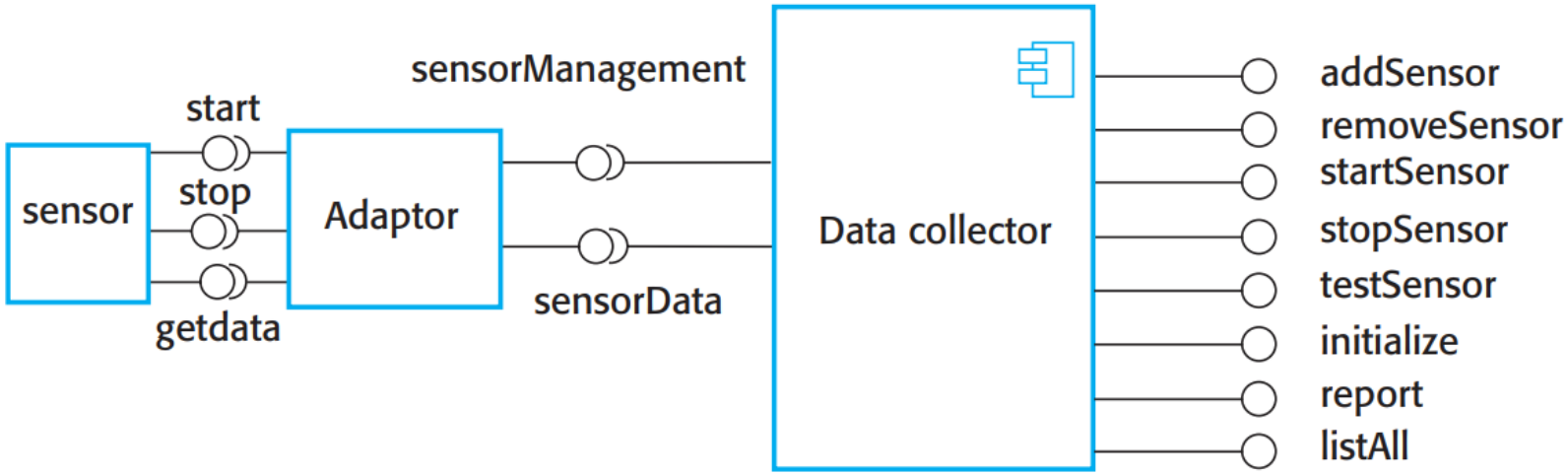
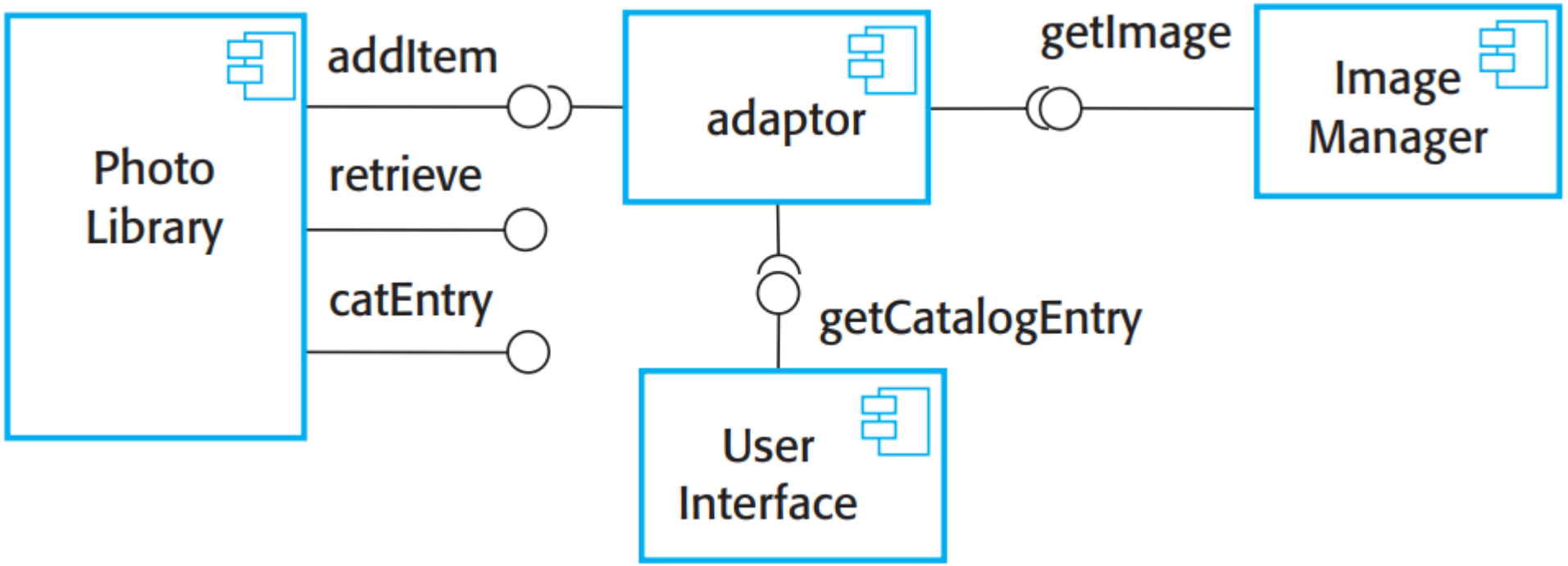


Photo library composition



Composition trade-offs

- เมื่อจะทำการรวบ component เราอาจพบข้อขัดแย้งระหว่าง functional และ non-functional requirement และข้อขัดแย้งระหว่างความต้องการการจัดส่งที่รวดเร็ว (rapid software development)และการพัฒนาระบบ (system evolution)
- เราต้องตัดสินใจหลายๆ อย่าง เช่น
  - องค์ประกอบใดของ component ที่มีประสิทธิภาพในการส่งมอบ functional requirement
  - องค์ประกอบของ component ใดบ้างที่สามารถเปลี่ยนแปลงได้ในอนาคต
  - อะไรคือคุณสมบัติฉุกเฉินของระบบที่นำ component มาประกอบขึ้น?

Key points

- วิศวกรรมซอฟต์แวร์แบบใช้ component เป็นแนวทางการพัฒนาซอฟต์แวร์ ที่เน้นการกำหนด และการสร้างด้วยวิธีการประกอบ component เข้าด้วยกันอย่างหลวม ๆ เพื่อสร้างระบบที่ใช้งานได้จริง
- Component คือหน่วยซอฟต์แวร์ที่มีการทำงานอย่างสมบูรณ์ โดยชุดของ public interface เป็นช่องทางในการสื่อสารกับระบบภายนอก
- Component สามารถประกอบกับส่วนอื่น ๆ ของระบบโดยไม่ต้องรู้วิธีการสร้างหรือการทำงานภายใน
- Component อาจถูกนำไปใช้เป็นส่วนหนึ่งของการปฏิบัติการที่รวมอยู่ในระบบหรือในรูปแบบ บริการภายนอกที่อ้างอิงจากภายในระบบ

Key points

- Component model กำหนดชุดมาตรฐานสำหรับ component รวมถึงมาตรฐานของ interface มาตรฐานการติดตั้ง (deployment) และมาตรฐานการใช้งาน
- ในระหว่างกระบวนการ CBSE จะมีกระบวนการย่อย ๆ สลับกันระหว่าง requirement engineering และ system design เราจะต้องชั่งน้ำหนักระหว่างการยึด requirement หรือปรับตาม services ที่มีใน components
- กระบวนการ component composition เปรียบได้กับการเชื่อมต่อวงจรไฟฟ้าให้สมบูรณ์ เพื่อสร้างระบบซอฟต์แวร์ รูปแบบของการทำ composition ประกอบด้วย sequential composition, hierarchical composition, และ additive composition.

Key points

- การประกอบระบบจาก component ที่มีอยู่ เราต้องเขียน code จำนวนหนึ่ง เพื่อประสานให้มันทำงานด้วยกันได้ เรียกว่า “glue code”
- ในการประกอบระบบ เราต้องพิจาณาเลือก component ที่รองรับ functional, non-function requirement และพิจารณาความยากง่ายในการทำ evolution ในอนาคตด้วยวิธีการเปลี่ยน component เอาไว้ด้วย