CPE3243 วิศวกรรมซอฟต์แวร์
(Software Engineering)

Piyavit Laung-Aram

Major of Computer Engineering

Faculty of Engineering

Ramkhamhaeng University, Thailand

กระบวนการซอฟต์แวร์(Software Processes)

และ

วัฏจักรการพัฒนาซอฟต์แวร์ (Software Development Lift Cycle-SDLC)

กระบวนการซอฟต์แวร์

(Software Processes)

กระบวนการซอฟต์แวร์

คำว่า **ซอฟต์แวร์** จะเป็นการระบุชุดของโปรแกรมคอมพิวเตอร์ ขั้นตอน และเอกสารที่เกี่ยวข้อง (ผังงาน คู่มือ ฯลฯ) ที่อธิบายโปรแกรมและวิธีการใช้งาน

กระบวนการซอฟต์แวร์ คือชุดของกิจกรรมและผลลัพธ์ที่เกี่ยวข้องซึ่งมีส่วนในการสร้างผลิตภัณฑ์ซอฟต์แวร์ วิศวกรซอฟต์แวร์ส่วนใหญ่ทำกิจกรรมเหล่านี้

กระบวนการซอฟต์แวร์ (ต่อ)

กิจกรรมหลักสี่ประการ ซึ่งเป็นกิจกรรมปกติสำหรับกระบวนการซอฟต์แวร์ทั้งหมด กิจกรรมเหล่านี้คือ

- **การระบุข้อมูลจำเพาะของซอฟต์แวร์-Software specifications**: ต้องกำหนดฟังก์ชันการทำงานของซอฟต์แวร์ ข้อมูลต่าง ๆ และข้อจำกัดในการทำงานของซอฟต์แวร์
- **การพัฒนาซอฟต์แวร์-Software development**: ต้องผลิตซอฟต์แวร์ให้ตรงตามข้อกำหนด
- **การตรวจสอบซอฟต์แวร์-Software validation**: ซอฟต์แวร์ต้องได้รับการตรวจสอบเพื่อให้แน่ใจว่าทำในสิ่งที่ลูกค้าต้องการ
- **การวิวัฒนาการของซอฟต์แวร์-Software evolution**: ซอฟต์แวร์ต้องพัฒนาเพื่อตอบสนองความต้องการของลูกค้าที่เปลี่ยนแปลงไป

แบบจำลองกระบวนการซอฟต์แวร์

(The Software Process Model)

แบบจำลองกระบวนการซอฟต์แวร์

แบบจำลองกระบวนการซอฟต์แวร์-เป็นตัวระบุคำนิยามของกระบวนการซอฟต์แวร์ ซึ่งนำเสนอจากมุมมองเฉพาะ โดยธรรมชาติแล้ว แบบจำลองนั้นจะทำให้เราเข้าใจกระบวนการซอฟต์แวร์ง่ายขึ้น

ดังนั้นแบบจำลองกระบวนการซอฟต์แวร์จึงเป็นนามธรรมของกระบวนการทำงานจริงที่แบบจำลองกำลังอธิบายอยู่

แบบจำลองกระบวนการซอฟต์แวร์กระกอบด้วยมีกิจกรรมต่าง ๆ ซึ่งเป็นส่วนหนึ่งของกระบวนการซอฟต์แวร์ ผลิตภัณฑ์ซอฟต์แวร์ และบทบาทของผู้ที่เกี่ยวข้องในด้านวิศวกรรมซอฟต์แวร์

แบบจำลองกระบวนการซอฟต์แวร์

ตัวอย่างบางส่วนของแบบจำลองกระบวนการซอฟต์แวร์ ได้แก่

- แบบจำลองเวิร์กโฟลว์ **(workflow model)**: แสดงชุดของกิจกรรมในกระบวนการพร้อมกับอินพุต เอาต์พุต และการอ้างอิงต่าง ๆ แบบจำลองนี้จะแสดงขั้นตอนการทำงานต่าง ๆ ของมนุษย์ในงานปกติที่มนุษย์ทำอยู่
- กระแสข้อมูลหรือแบบจำลองกิจกรรม(**dataflow or activity model)** : สิ่งนี้แสดงถึงกระบวนการต่าง ๆ (กระบวนการคือกลุ่มกิจกรรมต่าง ๆ) โดยจะแสดงให้เห็นว่าอินพุต เอาร์พุตของกระบวนการคืออะไร
- แบบจำลองบทบาท/การดำเนินการ(**role/action model**): หมายถึงบทบาทของบุคคลที่เกี่ยวข้องในกระบวนการซอฟต์แวร์และกิจกรรมที่พวกเขารับผิดชอบ

กระบวนทัศน์ของการพัฒนาซอฟต์แวร์ (software development paradigm)

1. แนวทางน้ำตก-The waterfall approach : การดำเนินการนี้ใช้กิจกรรมปกติสำหรับกระบวนการซอฟต์แวร์และสร้างเป็นขั้นตอนของกระบวนการที่แยกจากกัน เช่น ข้อกำหนดข้อกำหนด การออกแบบซอฟต์แวร์ การใช้งาน การทดสอบ และอื่นๆ หลังจากกำหนดแต่ละขั้นตอนแล้ว จะ "ออกจากระบบ" และการพัฒนาจะเข้าสู่ขั้นตอนถัดไป
2. การพัฒนาเชิงวิวัฒนาการ-Evolutionary development: วิธีการนี้จะสอดแทรกกิจกรรมของข้อมูลจำเพาะ การพัฒนา และการตรวจสอบความถูกต้อง ระบบเริ่มต้นได้รับการพัฒนาอย่างรวดเร็วจากข้อกำหนดหรือรายละเอียดที่เป็นนามธรรม

กระบวนทัศน์ของการพัฒนาซอฟต์แวร์ (software development paradigm)

3. การเปลี่ยนแปลงแบบเป็นทางการ-Formal transformation: วิธีนี้ขึ้นอยู่กับการสร้างข้อกำหนดด้วยระบบทางคณิตศาสตร์ และการแปลงข้อกำหนดนี้ โดยใช้วิธีการทางคณิตศาสตร์เป็นโปรแกรม การเปลี่ยนแปลงเหล่านี้คือ 'การรักษาความถูกต้อง' ซึ่งหมายความว่าคุณสามารถมั่นใจได้ว่าโปรแกรมที่พัฒนาแล้วมีคุณสมบัติตรงตามข้อกำหนด
4. การประกอบระบบจากส่วนประกอบที่นำกลับมาใช้ใหม่ได้-System assembly from reusable components: วิธีการนี้จะถือว่าส่วนต่างๆ ของระบบมีอยู่แล้ว กระบวนการพัฒนาระบบมีเป้าหมายในการผสานรวมชิ้นส่วนเหล่านี้แทนที่จะพัฒนาขึ้นใหม่ตั้งแต่เริ่มต้น

วิกฤตซอฟต์แวร์

(Software Crisis)

วิกฤตซอฟต์แวร์

1. ขนาด-Size: ซอฟต์แวร์มีราคาแพงและซับซ้อนมากขึ้นด้วยความซับซ้อนและความคาดหวังที่เพิ่มขึ้นจากซอฟต์แวร์ ตัวอย่างเช่น รหัสในสินค้าอุปโภคบริโภคเพิ่มขึ้นเป็นสองเท่าทุกๆ สองสามปี

2. คุณภาพ-Quality: ผลิตภัณฑ์ซอฟต์แวร์จำนวนมากมีคุณภาพต่ำ กล่าวคือ ผลิตภัณฑ์ซอฟต์แวร์มีข้อบกพร่องหลังจากนำไปใช้งาน เนื่องจากเทคนิคการทดสอบที่ไม่มีประสิทธิภาพ ตัวอย่างเช่น การทดสอบซอฟต์แวร์มักพบข้อผิดพลาด 25 รายการต่อโค้ด 1,000 บรรทัด

วิกฤตซอฟต์แวร์(ต่อ)

3. ต้นทุน-Cost: การพัฒนาซอฟต์แวร์มีค่าใช้จ่ายสูง กล่าวคือ ในแง่ของเวลาในการพัฒนาและเงินที่เกี่ยวข้อง ตัวอย่างเช่น การพัฒนาระบบอัตโนมัติขั้นสูงของ FAA มีค่าใช้จ่ายมากกว่า $700 ต่อบรรทัดของรหัส

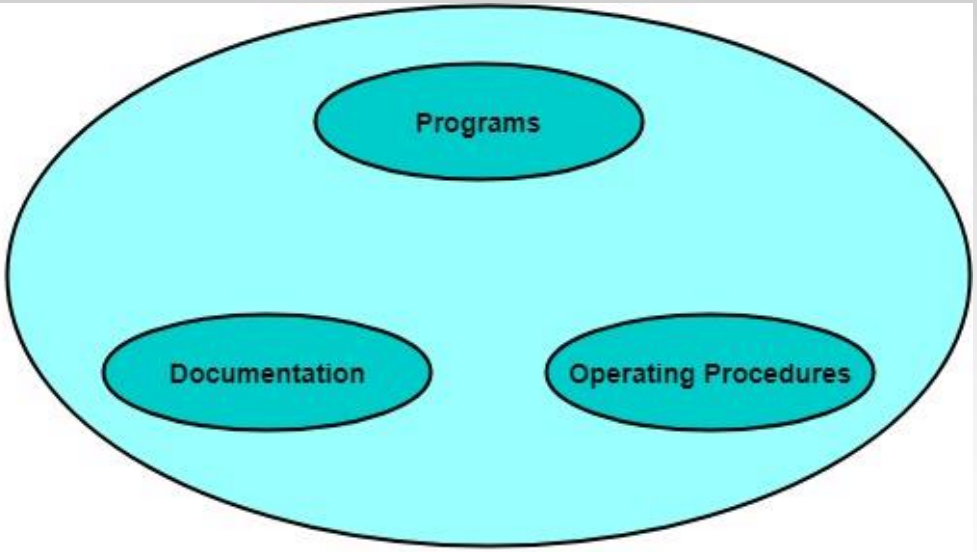
4. การจัดส่งล่าช้า-Delayed Delivery: กำหนดการเกินกำหนดที่ร้ายแรงเป็นเรื่องปกติ บ่อยครั้งที่ซอฟต์แวร์ใช้เวลานานกว่าเวลาโดยประมาณในการพัฒนา ซึ่งจะทำให้ต้นทุนสูงขึ้น ตัวอย่างเช่น หนึ่งในสี่โครงการพัฒนาขนาดใหญ่ไม่เคยเสร็จสมบูรณ์

โปรแกรมกับซอฟต์แวร์

(Program vs. Software)

โปรแกรมกับซอฟต์แวร์

ซอฟต์แวร์เป็นมากกว่าโปรแกรม โปรแกรมใด ๆ เป็นชุดย่อยของซอฟต์แวร์ และจะกลายเป็นซอฟต์แวร์ก็ต่อเมื่อมีการจัดทำคู่มือเอกสารและขั้นตอนการปฏิบัติงานมีสามองค์ประกอบของซอฟต์แวร์ดังแสดงในรูป



Software= Program + Documentation + Operating Procedures

Fig:Components of Software

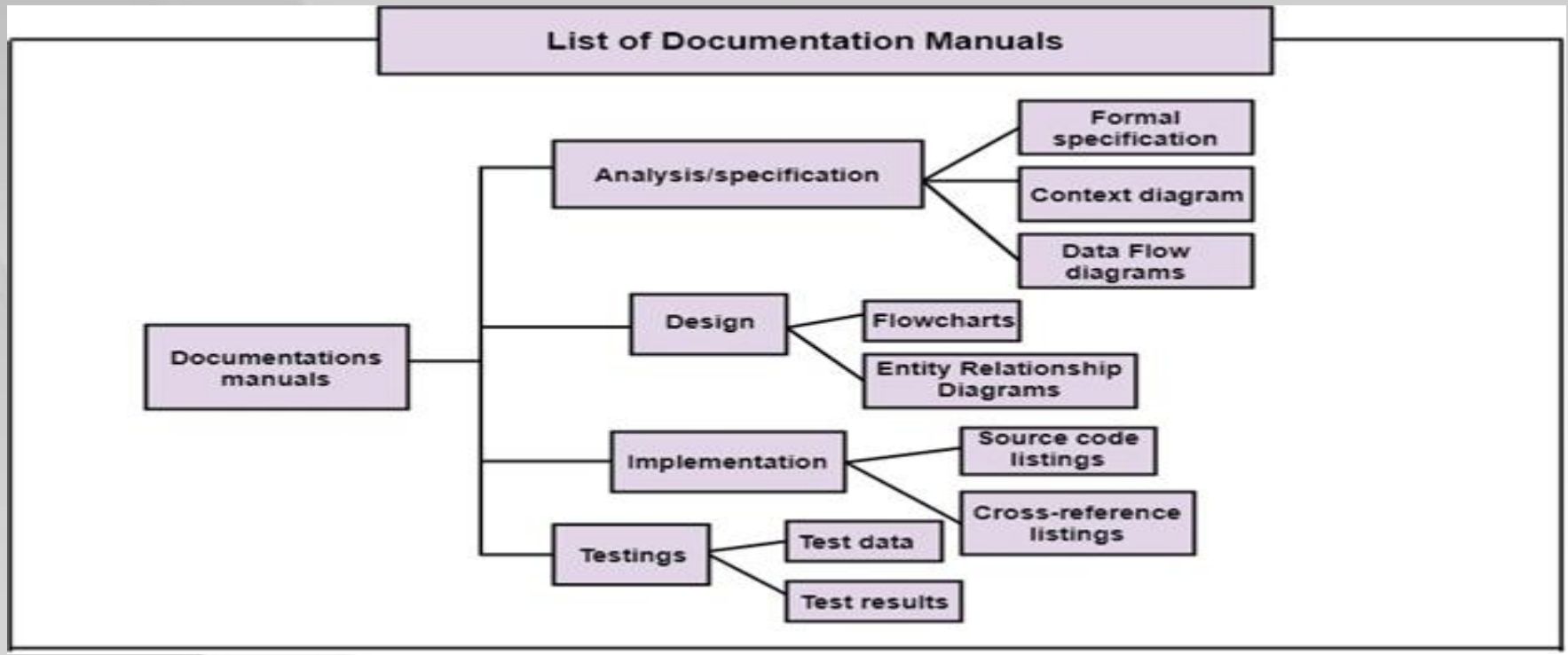
แหล่งที่มา https://www.javatpoint.com/

โปรแกรม (Program)

โปรแกรม: โปรแกรมคือการรวมกันของซอร์สโค้ดและรหัสอ็อปเจ็กต์

เอกสารประกอบ (Documentation)

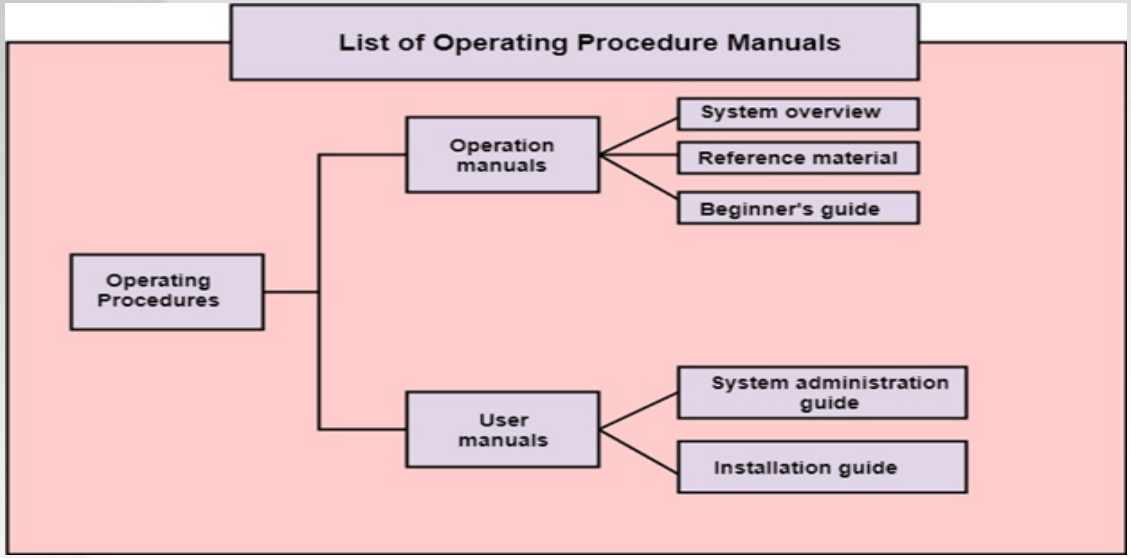
เอกสารประกอบ: เอกสารประกอบด้วยคู่มือประเภทต่างๆ ตัวอย่างของคู่มือเอกสาร ได้แก่ Data Flow Diagram, Flow Charts, ER Diagram เป็นต้น



แหล่งที่มา https://www.javatpoint.com/

ขั้นตอนการดำเนินงาน (Operating Procedures)

ขั้นตอนการดำเนินงาน: ขั้นตอนการปฏิบัติงานประกอบด้วยคำแนะนำในการตั้งค่าและใช้ระบบซอฟต์แวร์และคำแนะนำเกี่ยวกับวิธีการตอบสนองต่อความล้มเหลวของระบบ ตัวอย่างคู่มือขั้นตอนระบบปฏิบัติการ ได้แก่ คู่มือการติดตั้ง คู่มือสำหรับผู้เริ่มต้น คู่มืออ้างอิง คู่มือการดูแลระบบ ฯลฯ



แหล่งที่มา https://www.javatpoint.com/

วัฏจักรการพัฒนาซอฟต์แวร์ (Software Development Lift Cycle-SDLC)

แบบจำลองกระบวนการซอฟต์แวร์

แบบจำลองวงจรชีวิตของซอฟต์แวร์-software life cycle model (เรียกอีกอย่างว่า แบบจำลองกระบวนการ-process model) คือการแสดงภาพและแผนภาพของวงจรชีวิตของซอฟต์แวร์ แบบจำลองวงจรชีวิตแสดงวิธีการต่าง ๆ ทั้งหมดที่จำเป็นในการสร้างผลิตภัณฑ์ซอฟต์แวร์ ผ่านขั้นตอนของวงจรชีวิตของซอฟต์แวร์

กล่าวอีกนัยหนึ่ง แบบจำลองวงจรชีวิตของซอฟต์แวร์จะจับคู่กิจกรรมต่างๆ ที่ดำเนินการกับผลิตภัณฑ์ซอฟต์แวร์ตั้งแต่เริ่มต้นจนสิ้นสุด แบบจำลองวัฏจักรชีวิตของซอฟต์แวร์มีความแตกต่างกันได้ตามแนวทางในการพัฒนาซอฟต์แวร์แต่ละแนวทาง ดังนั้นจึงไม่มีองค์ประกอบที่ตายตัว กิจกรรมที่จำเป็นจะมีอยู่ในแบบจำลองวงจรชีวิตทั้งหมด แม้ว่าการดำเนินการอาจดำเนินการในลำดับที่แตกต่างกันในแบบจำลองวงจรชีวิตของซอฟต์แวร์ที่แตกต่างกัน

ความจำเป็นของ SDLC

ทีมพัฒนาต้องกำหนดรูปแบบวงจรชีวิตที่เหมาะสมสำหรับการพัฒนาซอฟต์แวร์

หากไม่ใช้แบบจำลองวงจรชีวิตที่แน่นอน การพัฒนาผลิตภัณฑ์ซอฟต์แวร์จะไม่เป็นไปตามแนวทางอย่างเป็นระบบและมีระเบียบวินัย เมื่อทีมกำลังพัฒนาผลิตภัณฑ์ซอฟต์แวร์ ตัวแทนทีมจะต้องมีความเข้าใจที่ชัดเจนเกี่ยวกับเวลาและสิ่งที่ต้องทำ มิฉะนั้นจะชี้ให้เห็นถึงความโกลาหลและความล้มเหลวของโครงการ ปัญหานี้สามารถกำหนดได้โดยใช้ตัวอย่าง ดังนี้

ความจำเป็นของ SDLC (ต่อ)

สมมุติว่าปัญหาด้านการพัฒนาซอฟต์แวร์แบ่งออกเป็นส่วนต่างๆ และส่วนที่กำหนดให้กับสมาชิกในทีม จากนั้นเป็นต้นมา สมมติว่าตัวแทนทีมได้รับอนุญาตให้มีอิสระในการพัฒนา มีบทบาทที่ได้รับมอบหมายให้พวกเขาในแบบที่พวกเขาต้องการ เป็นไปได้ว่าตัวแทนคนหนึ่งอาจเริ่มเขียนโค้ดสำหรับส่วนของตน อีกคนอาจเลือกเตรียมเอกสารการทดสอบก่อน และวิศวกรคนอื่นๆ อาจเริ่มต้นด้วยขั้นตอนการออกแบบของบทบาทที่ได้รับมอบหมาย นี่จะเป็นหนึ่งในวิธีการที่สมบูรณ์แบบสำหรับความล้มเหลวของโครงการ

ความต้องการของ SDLC (ต่อ)

แบบจำลองวงจรชีวิตของซอฟต์แวร์จะอธิบายเกณฑ์การเข้าและออกสำหรับแต่ละเฟสของขั้นตอนในการพัฒนา ขั้นตอนจะสามารถเริ่มต้นได้ก็ต่อเมื่อเป็นไปตามเกณฑ์การเข้าสู่ขั้นตอนนั้น ๆ ดังนั้นหากไม่มีแบบจำลองวงจรชีวิตซอฟต์แวร์ เกณฑ์การเข้าและออกจากขั้นตอนจะไม่เป็นที่รู้จัก หากไม่มีแบบจำลองวงจรชีวิตซอฟต์แวร์ ผู้จัดการโครงการซอฟต์แวร์จะติดตามความคืบหน้าของโครงการได้ยาก

SDLC

SDLC แสดงถึงกระบวนการพัฒนาซอฟต์แวร์ กรอบงาน SDLC ประกอบด้วยขั้นตอนต่อไปนี้

แหล่งที่มา https://www.javatpoint.com/

ขั้นตอนของ SDLC มีดังนี้

ขั้นตอนที่ 1: การวางแผนและการวิเคราะห์ความต้องการ-Planning and requirement analysis

ขั้นตอนที่ 2: การะบุข้อกำหนด-Defining Requirements

ขั้นตอนที่ 3: การออกแบบซอฟต์แวร์-Designing the Software

ขั้นตอนที่ 4: การพัฒนาโครงการ-Developing the project

ขั้นตอนที่ 5: การทดสอบ-Testing

ขั้นตอนที่ 6: การติดตั้งหรือนำไปใช้งาน-Deployment

ขั้นตอนที่ 7: การบำรุงรักษา -Maintenance

ขั้นตอนที่ 1:การวางแผนและการวิเคราะห์ความต้องการ-Planning and requirement analysis

การวิเคราะห์ความต้องการเป็นขั้นตอนที่สำคัญและจำเป็นที่สุดใน SDLC

สมาชิกอาวุโสหรือผู้มีประสบการณ์ของทีมจะดำเนินการรวบรวมข้อมูลข้อมูลจากผู้มีส่วนได้ส่วนเสียทั้งหมดและผู้เชี่ยวชาญในโดเมนนั้น ๆ หรือ SMEs ในอุตสาหกรรม

การวางแผนสำหรับข้อกำหนดด้านการประกันคุณภาพและการระบุความเสี่ยงที่เกี่ยวข้องกับโครงการจะดำเนินการในขั้นตอนนี้ด้วย นักวิเคราะห์ธุรกิจและผู้จัดโครงการจัดประชุมกับลูกค้าเพื่อรวบรวมข้อมูลทั้งหมด เช่น สิ่งที่ลูกค้าต้องการสร้าง ใครจะเป็นผู้ใช้ปลายทาง วัตถุประสงค์ของผลิตภัณฑ์คืออะไร ก่อนสร้างผลิตภัณฑ์จำเป็นต้องมีความเข้าใจหลักหรือความรู้เกี่ยวกับผลิตภัณฑ์

ขั้นตอนที่ 1: การวางแผนและการวิเคราะห์ความต้องการ- Planning and requirement analysis(ต่อ)

ตัวอย่างเช่น ลูกค้าต้องการมีแอปพลิเคชันที่เกี่ยวข้องกับธุรกรรมการเงิน ในวิธีนี้ ข้อกำหนดจะต้องแม่นยำ เช่น การดำเนินการประเภทใด จะทำอย่างไร สกุลเงินใดที่จะทำ ฯลฯ

เมื่อฟังก์ชันที่จำเป็นเสร็จสิ้น การวิเคราะห์จะสมบูรณ์ด้วยการตรวจสอบความเป็นไปได้ในการเติบโตของผลิตภัณฑ์ ในกรณีที่มีความกำกวม จะมีการตั้งสัญญาณเพื่อการอภิปรายต่อไป

เมื่อเข้าใจข้อกำหนดแล้ว เอกสาร SRS (Software Requirement Specification-ข้อกำหนดความต้องการของซอฟต์แวร์) จะถูกสร้างขึ้น นักพัฒนาซอฟต์แวร์ควรปฏิบัติตามเอกสารนี้อย่างละเอียดและควรได้รับการตรวจสอบโดยลูกค้าเพื่อใช้อ้างอิงในอนาคต

ขั้นตอนที่ 2: การระบุข้อกำหนดความต้องการ

Defining Requirements

เมื่อการวิเคราะห์ความต้องการเสร็จสิ้น ขั้นตอนต่อไปคือการจัดทำเอกสารข้อกำหนดของซอฟต์แวร์ และทำให้ได้รับการยอมรับจากผู้มีส่วนได้ส่วนเสียของโครงการ

สามารถทำได้ผ่าน "SRS" - เอกสารข้อกำหนดของซอฟต์แวร์ ซึ่งมีข้อกำหนดของผลิตภัณฑ์ทั้งหมดที่จะสร้างและพัฒนาในระหว่างวงจรชีวิตโครงการ

ขั้นตอนที่ 3:การออกแบบซอฟต์แวร์-Designing the Software

ขั้นต่อไปกำลังจะนำความรู้ทั้งหมดเกี่ยวกับความต้องการ การวิเคราะห์ และการออกแบบของโครงการซอฟต์แวร์ลงมาใช้ ระยะนี้เป็นผลผลิตของขั้นตอนที่ 1 และ 2 คือ ข้อมูลจากลูกค้าและการรวบรวมความต้องการต่าง ๆ ทั้งทางด้านข้อมูลและเทคนิค

ขั้นตอนที่ 4:การพัฒนาโครงการ-Developing the project

ในขั้นตอนนี้ของ SDLC การพัฒนาจริงเริ่มต้นขึ้นและการเขียนโปรแกรมก็ถูกสร้างขึ้น การดำเนินการออกแบบเริ่มต้นเกี่ยวกับการเขียนโค้ด นักพัฒนาซอฟต์แวร์ต้องปฏิบัติตามแนวทางการเขียนโค้ดที่กำหนดไว้ในขั้นตอนการดำเนินงาน โดยเครื่องมือการจัดการและการเขียนโปรแกรม เช่น คอมไพเลอร์ อินเทอร์พรีตเทอร์ ดีบักเกอร์ ฯลฯ เพื่อใช้ในการพัฒนาและใช้งานโค้ด

ขั้นตอนที่ 5:การทดสอบ-Testing

หลังจากสร้างโค้ดแล้ว จะมีการทดสอบกับข้อกำหนดเพื่อให้แน่ใจว่าผลิตภัณฑ์สามารถแก้ไขความต้องการที่ได้รับการแก้ไขและรวบรวมในระหว่างขั้นตอนข้อกำหนดในระหว่างขั้นตอนนี้ การทดสอบหน่วยย่อย การทดสอบการรวม การทดสอบระบบ การทดสอบการยอมรับจะเสร็จสิ้น

ขั้นตอนที่ 6: การติดตั้งหรือนำไปใช้งาน-Deployment

เมื่อซอฟต์แวร์ได้รับการรับรอง และไม่พบจุดบกพร่องหรือข้อผิดพลาดต่าง ๆ แล้ว ซอฟต์แวร์จะถูกนำไปใช้งาน

จากนั้นจะตามมาด้วยการประเมินการใช้งาน ซอฟต์แวร์จะถูกติดตั้งหรือนำไปใช้งานตามแผนที่ออกแบบไว้ โดยจะระบุอยู่ในคู่มือการติดตั้งหรือนำไปใช้งาน

หลังจากซอฟต์แวร์ถูกนำไปติดตั้งหรือนำไปใช้งานแล้ว การบำรุงรักษาก็จะเริ่มขึ้น

ขั้นตอนที่ 7:การบำรุงรักษา -Maintenance

เมื่อลูกค้าเริ่มใช้ระบบที่พัฒนาแล้ว ปัญหาที่แท้จริงจะเกิดขึ้นและความต้องการที่จะแก้ไขปรับปรุงซอฟต์แวร์ตามที่ต้องการจะตามมา เรียกขั้นตอนนี้ว่า การบำรุงรักษา

กระบวนงานนี้เป็นกระบวนการที่ต้องใช้ความระมัดระวังเป็นพิเศษสำหรับผลิตภัณฑ์ที่พัฒนาขึ้น เนื่องจากจะเป็นกระบวนการที่มีค่าใช้จ่ายต่ำหรือสูงขึ้นกับการวางแผนเริ่มต้นโครงการตั้งแต่ต้น

โมเดลวัฏจักรการพัฒนาซอฟต์แวร์

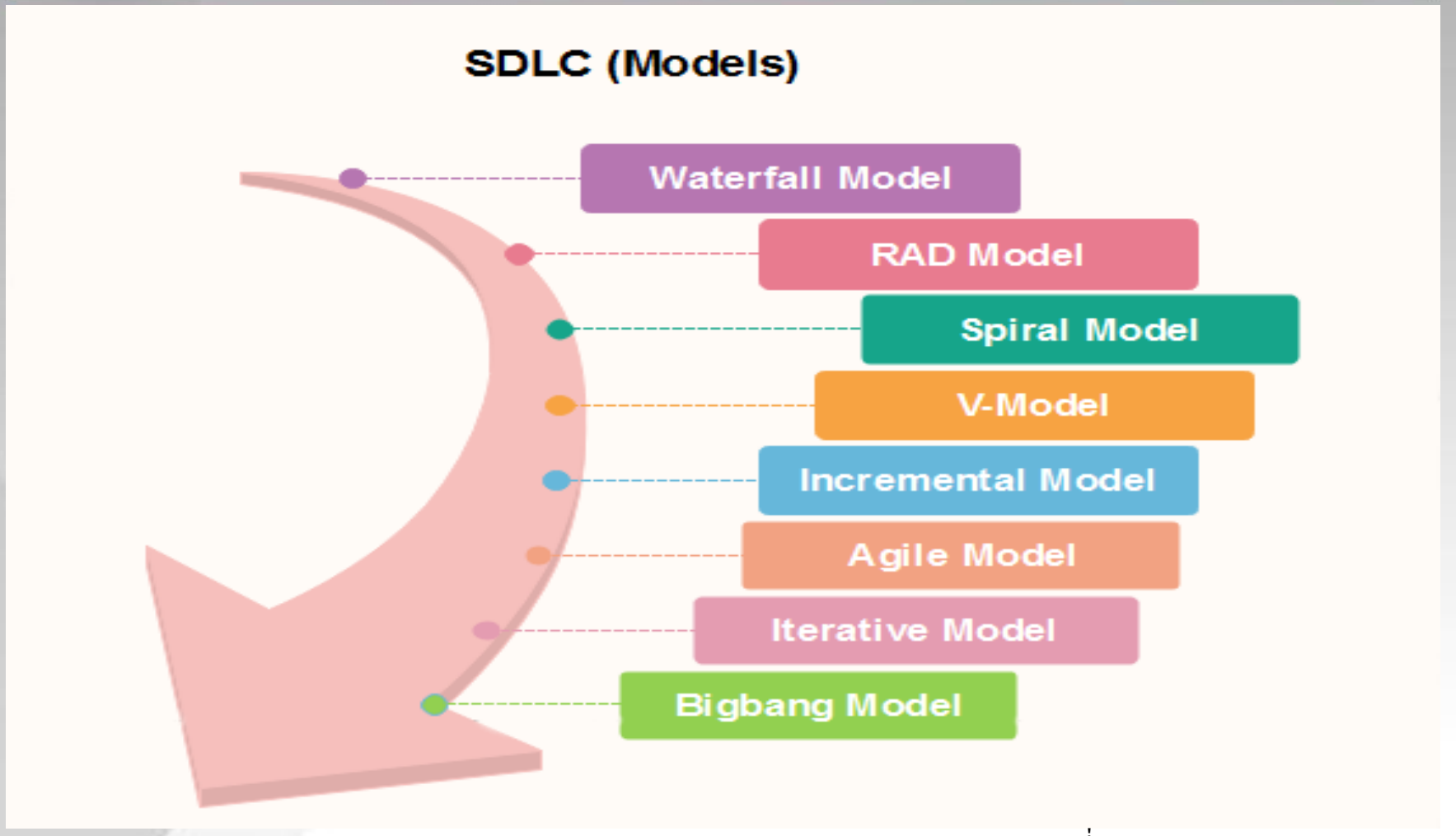
(Software Development Lift Cycle Models-SDLC Models)

โมเดล SDLC

วัฏจักรการพัฒนาซอฟต์แวร์ (SDLC) เป็นแบบจำลองที่ใช้ในการจัดการโครงการที่กำหนดขั้นตอนที่รวมอยู่ในโครงการพัฒนาซอฟต์แวร์ ตั้งแต่การศึกษาความเป็นไปได้เบื้องต้นไปจนถึงการบำรุงรักษาซอฟต์แวร์ ที่เสร็จสมบูรณ์

การระบุโมเดลและออกแบบรูปแบบวงจรชีวิตการพัฒนาซอฟต์แวร์ที่แตกต่างกัน ขึ้นกับรูปแบบซอฟต์แวร์ที่พัฒนา เพื่อให้มั่นใจว่าการพัฒนาซอฟต์แวร์จะประสบความสำเร็จตามที่วางแผนไว้

โมเดล SDLC (ต่อ)



แหล่งที่มา https://www.javatpoint.com/

แบบจำลองน้ำตก(Waterfall Model)

แบบจำลองน้ำตก(Waterfall Model)

แบบจำลองน้ำตกเป็นโมเดลการพัฒนาซอฟต์แวร์อย่างต่อเนื่อง โดยที่การพัฒนาจะถูกมองว่าเหมือนน้ำไหลลงมาอย่างต่อเนื่อง (เช่น น้ำตก) ผ่านขั้นตอนต่างๆ ของการวิเคราะห์ความต้องการ การออกแบบ การนำไปใช้ การทดสอบ (การตรวจสอบความถูกต้อง) การผสานรวม และการบำรุงรักษาการเรียงลำดับกิจกรรมเชิงเส้นมีผลที่ตามมาที่สำคัญบางประการ ขั้นแรก เพื่อระบุจุดสิ้นสุดของระยะและจุดเริ่มต้นของขั้นตอนถัดไป ต้องใช้เทคนิคการรับรองบางอย่างเมื่อสิ้นสุดแต่ละขั้นตอน การตรวจสอบและรับรองความถูกต้องบางอย่างมักจะทำสิ่งนี้หมายความว่าจะทำให้แน่ใจว่าเอาต์พุตของสเตจสอดคล้องกับอินพุต (ซึ่งเป็นเอาต์พุตของขั้นตอนก่อนหน้า) และเอาต์พุตของสเตจนั้นสอดคล้องกับข้อกำหนดโดยรวมของระบบ

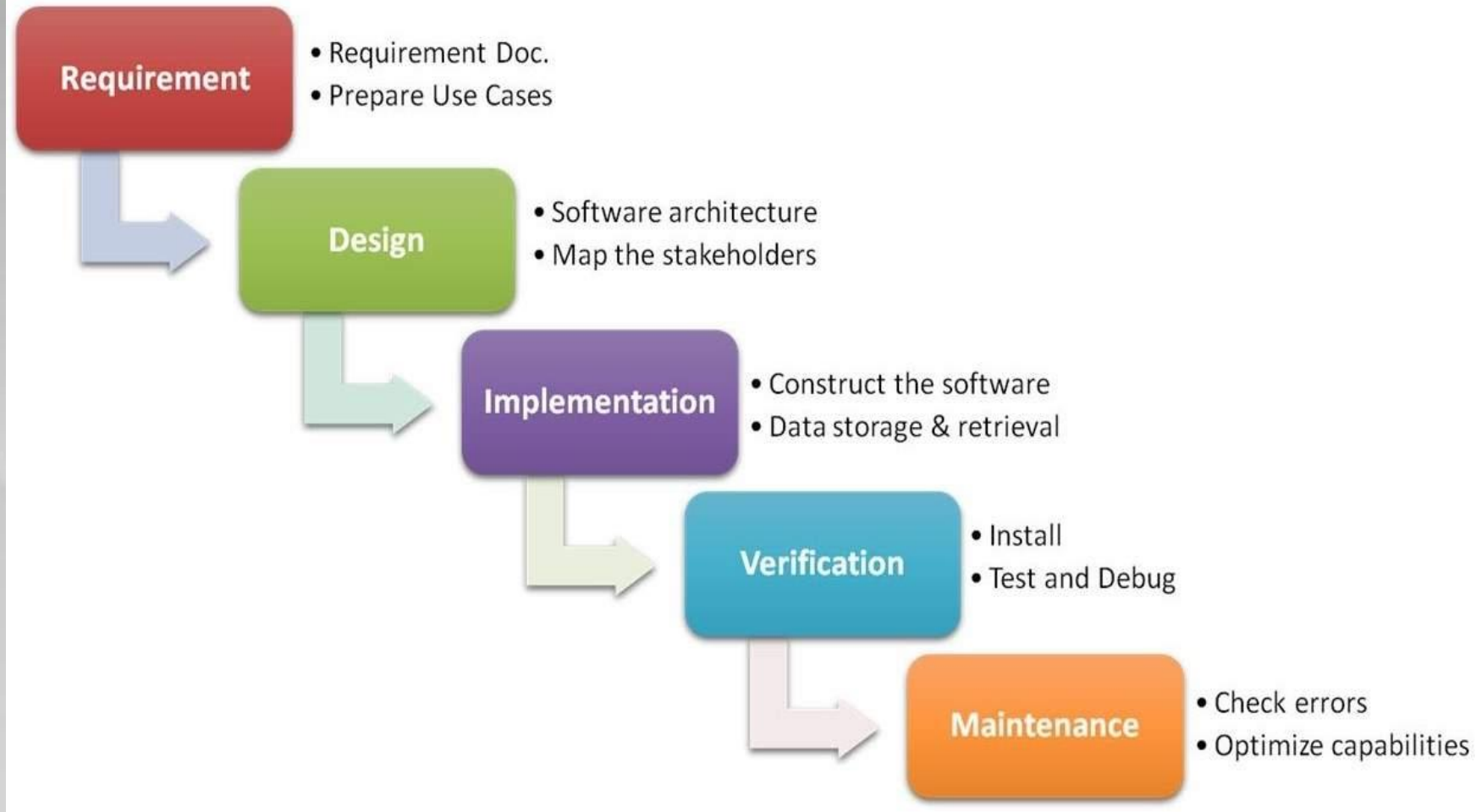
แบบจำลองน้ำตก(Waterfall Model) (ต่อ)

Winston Royce ได้เปิดตัว Waterfall Model ในปี 1970 โมเดลนี้มีห้าขั้นตอน:

- การวิเคราะห์ข้อกำหนดและข้อกำหนด
- การออกแบบ
- การพัฒนาและการใช้งาน
- การทดสอบระบบ
- การบำรุงรักษา

ทำตามขั้นตอนตามลำดับนี้เสมอและไม่ทับซ้อนกัน ผู้พัฒนาต้องดำเนินการให้เสร็จสิ้นทุกขั้นตอนก่อนที่จะเริ่มเฟสต่อไป แบบจำลองนี้มีชื่อว่า "Waterfall Model" เนื่องจากการแสดงแผนภาพคล้ายกับน้ำตก

แบบจำลองน้ำตก(Waterfall Model) (ต่อ)



แหล่งที่มา https://www.ukessays/essays/

เมื่อใดจึงควร แบบจำลองน้ำตก(Waterfall Model) ?

สถานการณ์ที่การใช้แบบจำลองน้ำตกเหมาะสมที่สุด ได้แก่ :

- เมื่อความต้องการคงที่และไม่เปลี่ยนแปลงอย่างสม่ำเสมอ
- เมื่อเป็นโครงการระยะสั้น
- เมื่อโครงการสามารถควบคุมปัญหา และข้อผิดพลาดต่าง ๆ ได้
- เมื่อเครื่องมือและเทคโนโลยีที่ใช้มีความสม่ำเสมอและไม่เปลี่ยนแปลง
- เมื่อทรัพยากรถูกเตรียมมาอย่างดีและพร้อมใช้งาน

ข้อดี แบบจำลองน้ำตก(Waterfall Model)

- แบบจำลองนี้ง่ายต่อการใช้งาน ใช้ทรัพยากรน้อย
- ข้อกำหนดนั้นเรียบง่ายและประกาศไว้อย่างชัดเจน จะไม่มีการเปลี่ยนแปลงข้อกำหนดในระหว่างการพัฒนาโครงการทั้งหมด
- จุดเริ่มต้นและจุดสิ้นสุดสำหรับแต่ละเฟสกำหนดไว้อย่างตายตัว ทำให้ง่ายต่อการควบคุมความคืบหน้าในการพัฒนา
- สามารถกำหนดวันที่วางจำหน่ายสำหรับผลิตภัณฑ์ที่สมบูรณ์รวมถึงต้นทุนขั้นสุดท้ายได้ก่อนการพัฒนา
- ช่วยให้ควบคุมได้ง่ายและชัดเจนสำหรับลูกค้าเนื่องจากระบบการรายงานที่เข้มงวด

ข้อเสีย แบบจำลองน้ำตก(Waterfall Model)

- ในแบบจำลองนี้ ปัจจัยเสี่ยงจะสูงหลายประเด็น ดังนั้นแบบจำลองนี้จึงไม่เหมาะสำหรับโครงการที่มีนัยสำคัญและซับซ้อน
- แบบจำลองนี้ไม่สามารถเปลี่ยนแปลงข้อกำหนดในระหว่างการพัฒนา
- การย้อนกลับเข้าสู่เฟสก่อนหน้าทำได้ยาก ตัวอย่างเช่น หากตอนนี้แอปพลิเคชันได้เปลี่ยนไปสู่ขั้นตอนการเข้ารหัส และมีการเปลี่ยนแปลงข้อกำหนด การย้อนกลับและเปลี่ยนแปลงนั้นทำได้ยาก
- เนื่องจากการทดสอบทำในขั้นตอนท้าย ๆ ทำให้ไม่สามารถระบุสิ่งที่ต้องเผชิญและความเสี่ยงในเฟสก่อนหน้าได้ ดังนั้นจึงเป็นการยากที่จะเตรียมกลยุทธ์ในการลดความเสี่ยงในการพัฒนาซอฟต์แวร์

แบบจำลอง RAD

(Rapid Application Development Model-RAD Model)

แบบจำลอง RAD (RAD Model)

แบบจำลอง RAD หรือ Rapid Application Development เป็นการนำแบบจำลองน้ำตกมาใช้ มุ่งเป้าไปที่การพัฒนาซอฟต์แวร์ในระยะเวลาอันสั้น โมเดล RAD ขึ้นอยู่กับแนวคิดที่ว่าระบบที่ดีขึ้นสามารถพัฒนาได้ในเวลาน้อยลงโดยใช้โฟกัสกรุ๊ปเพื่อรวบรวมความต้องการของระบบ

- การสร้างแบบจำลองทางธุรกิจ
- การสร้างแบบจำลองข้อมูล
- การสร้างแบบจำลองกระบวนการ
- การสร้างแอปพลิเคชัน
- การทดสอบและการหมุนเวียน

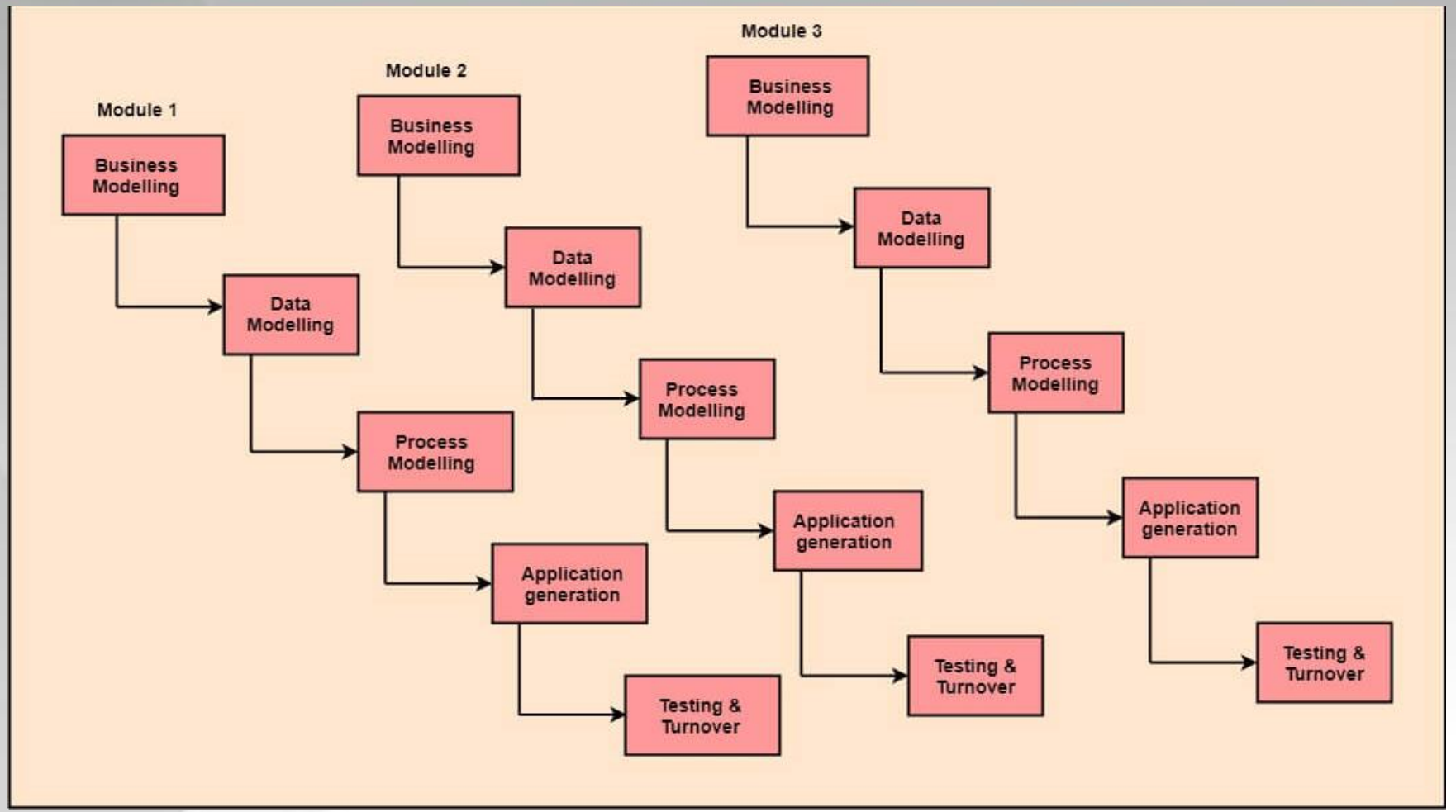
แบบจำลอง RAD (RAD Model) (ต่อ)

RAD เป็นแบบจำลองกระบวนการพัฒนาซอฟต์แวร์แบบลำดับเชิงเส้นที่เน้นวงจรการพัฒนาที่รัดกุมโดยใช้วิธีการแบบอิงองค์ประกอบ หากข้อกำหนดมีความเข้าใจและอธิบายเป็นอย่างดี และขอบเขตของโครงการมีข้อจำกัด กระบวนการ RAD จะช่วยให้ทีมพัฒนาสามารถสร้างระบบที่ทำงานได้อย่างสมบูรณ์ภายในระยะเวลาที่กำหนด

RAD (Rapid Application Development) เป็นแนวคิดที่ว่าผลิตภัณฑ์สามารถพัฒนาได้เร็วและมีคุณภาพสูงขึ้น โดยดำเนินการดังนี้

- การรวบรวมความต้องการโดยใช้การประชุมเชิงปฏิบัติการหรือโฟกัสกรุ๊ป
- การสร้างต้นแบบและการทดสอบการออกแบบโดยผู้ใช้ซ้ำๆ ในระยะเริ่มต้น
- การนำส่วนประกอบซอฟต์แวร์กลับมาใช้ใหม่
- มีการสร้างกำหนดการที่รัดกุม ในการปรับปรุงการออกแบบสำหรับผลิตภัณฑ์รุ่นถัดไป
- การสื่อสารในทีมทำงานมีความเป็นทางการน้อยลง

แบบจำลอง RAD (RAD Model) (ต่อ)



แหล่งที่มา https://www.javatpoint.com/

แบบจำลอง RAD (RAD Model) (ต่อ)

ขั้นตอนต่างๆ ของแบบจำลอง RAD มีดังนี้:

1.การสร้างแบบจำลองทางธุรกิจ-Business Modelling: การไหลของข้อมูลระหว่างฟังก์ชันทางธุรกิจถูกกำหนดโดยการตอบคำถาม เช่น ข้อมูลใดบ้างที่ขับเคลื่อนกระบวนการทางธุรกิจ ข้อมูลที่สร้างขึ้น ใครเป็นคนสร้าง ข้อมูลไปไหน ใครเป็นผู้ดำเนินการ และอื่นๆ

2. การสร้างแบบจำลองข้อมูล-Data Modelling: ข้อมูลที่รวบรวมจากการสร้างแบบจำลองธุรกิจได้รับรวบรวมเป็นเซตของข้อมูลหรือเอนทิตี ที่จำเป็นต่อการสนับสนุนธุรกิจ มีการระบุเอนทิตี และมีการกำหนดความสัมพันธ์ระหว่างเอนทิตี

แบบจำลอง RAD (RAD Model) (ต่อ)

3. การสร้างแบบจำลองกระบวนการ-Process Modelling: เอนทิตีที่กำหนดไว้ในขั้นตอนการสร้างแบบจำลองข้อมูลจะถูกแปลงเพื่อให้ได้กระแสข้อมูลที่จำเป็นในการใช้งานฟังก์ชันทางธุรกิจ

4. การสร้างแอปพลิเคชัน-Application Generation: ใช้เครื่องมืออัตโนมัติใช้เพื่ออำนวยความสะดวกในการสร้างซอฟต์แวร์

5. การทดสอบและการหมุนเวียน-Testing & Turnover: ส่วนประกอบการเขียนโปรแกรมจำนวนมากได้รับการทดสอบแล้ว สามารถนำกลับมาหมุนเวียนใช้ซ้ำได้เลย ซึ่งจะช่วยลดเวลาการทดสอบโดยรวม แต่ส่วนใหม่จะต้องได้รับการทดสอบก่อน และส่วนต่อประสานทั้งหมดจะต้องได้รับการใช้งานอย่างเต็มที่

เมื่อใดควรใช้*แบบจำลอง* RAD?

- เมื่อระบบควรต้องสร้างโปรเจ็กต์แบบโมดูลาร์ในระยะเวลาอันสั้น (2-3 เดือน)
- เมื่อความต้องการเป็นที่ทราบกันดี
- เมื่อความเสี่ยงทางเทคนิคมีจำกัด
- เมื่อมีความจำเป็นต้องสร้างระบบที่แยกส่วนได้ภายในระยะเวลา 2-3 เดือน
- ควรใช้เฉพาะในกรณีที่มีงบประมาณอนุญาตให้ใช้เครื่องมือสร้างโค้ดอัตโนมัติ

ข้อดีของ RAD Model

- มีความยืดหยุ่นในการเปลี่ยนแปลง
- การเปลี่ยนแปลงเป็นที่ยอมรับในแต่ละเฟส
- แต่ละเฟสใน RAD จะนำฟังก์ชันที่มีลำดับความสำคัญสูงสุดมาสู่ลูกค้า
- ลดเวลาในการพัฒนา
- ช่วยเพิ่มคุณสมบัติการนำกลับมาใช้ใหม่ได้

ข้อเสียของแบบจำลอง RAD

- ต้องใช้นักออกแบบที่มีทักษะสูง
- ไม่ใช่ทุกแอปพลิเคชันที่สามารถใช้แบบจำลอง RAD ในการพัฒนาได้
- สำหรับโครงการขนาดเล็ก เราไม่สามารถใช้แบบจำลอง RAD ได้
- มีความเสี่ยงทางเทคนิคสูง จึงไม่เหมาะ
- ต้องการการมีส่วนร่วมของผู้ใช้ในการพัฒนาเป็นอย่างมาก

แบบจำลอง Spiral

(Spiral Model)

แบบจำลอง Spiral

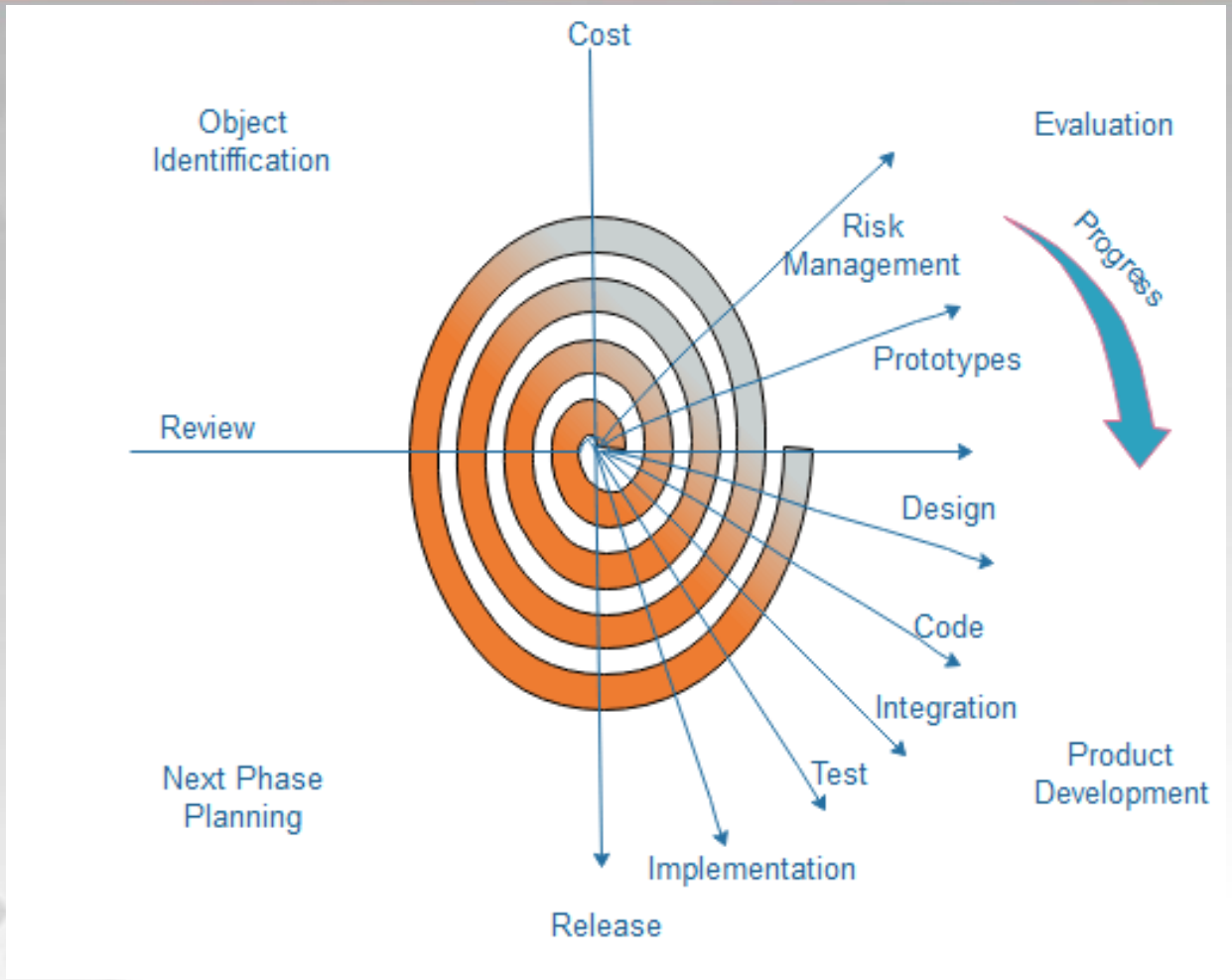
แบบจำลอง Spiral หรือ แบบจำลองแบบ เกลียว เป็นแบบจำลองกระบวนการที่ขับเคลื่อนโดยความเสี่ยง แบบจำลอง SDLC นี้ช่วยให้กลุ่มนำองค์ประกอบของแบบจำลองกระบวนการตั้งแต่หนึ่งแบบจำลองขึ้นไปมาใช้ เช่น น้ำตก ส่วนที่เพิ่มขึ้นของแบบจำลองน้ำตก เป็นต้น เทคนิคเกลียวเป็นการผสมผสานระหว่างการสร้างต้นแบบอย่างรวดเร็วและการทำงานพร้อมกันในกิจกรรมการออกแบบและพัฒนาแต่ละรอบในวงก้นหอย

เริ่มต้นด้วยการระบุวัตถุประสงค์ของวัฏจักรนั้น ทางเลือกต่างๆ ที่เป็นไปได้สำหรับการบรรลุเป้าหมาย และข้อจำกัดที่มีอยู่ นี่คือจตุภาคแรกของวัฏจักร (จตุภาคบนซ้าย)

ขั้นตอนต่อไปของวงจรคือการประเมินทางเลือกต่างๆ เหล่านี้ตามวัตถุประสงค์และข้อจำกัด จุดเน้นของการประเมินในขั้นตอนนี้ขึ้นอยู่กับการรับรู้ความเสี่ยงของโครงการ

ขั้นตอนต่อไปคือการพัฒนากลยุทธ์ที่แก้ไขความไม่แน่นอนและความเสี่ยง ขั้นตอนนี้อาจเกี่ยวข้องกับกิจกรรมต่างๆ เช่น การเปรียบเทียบ การจำลอง และการสร้างต้นแบบ

แบบจำลอง Spiral (ต่อ)



แหล่งที่มา https://www.javatpoint.com/

แบบจำลอง Spiral *แบ่งออกเป็นสี่ส่วน*

การตั้งค่าวัตถุประสงค์- Objective setting : แต่ละรอบในเกลียวเริ่มต้นด้วยการระบุวัตถุประสงค์สำหรับวงจรนั้น ทางเลือกต่างๆ ที่เป็นไปได้สำหรับการบรรลุเป้าหมายและข้อจำกัดที่มีอยู่

การประเมินและการลดความเสี่ยง- Risk Assessment and reduction : ขั้นตอนต่อไปของวงจรคือการคำนวณทางเลือกต่างๆ เหล่านี้ตามเป้าหมายและข้อจำกัด จุดเน้นของการประเมินในขั้นตอนนี้อยู่ที่การรับรู้ความเสี่ยงของโครงการ

การพัฒนาและการตรวจสอบความถูกต้อง- Development and validation : ขั้นตอนต่อไปคือการพัฒนากลยุทธ์ที่แก้ไขความไม่แน่นอนและความเสี่ยง กระบวนการนี้อาจรวมถึงกิจกรรมต่างๆ เช่น การเปรียบเทียบ การจำลอง และการสร้างต้นแบบ

แบบจำลอง Spiral *แบ่งออกเป็นสี่ส่วน(ต่อ)*

การวางแผน-Planning: ขั้นตอนต่อไปก็คือการวางแผน โปรเจ็กต์นี้ได้รับการตรวจสอบแล้ว และมีตัวเลือกว่าจะดำเนินการต่อไปหรือไม่ หากตั้งใจที่จะรักษาไว้ แผนงานสำหรับขั้นตอนต่อไปของโครงการจะถูกร่างขึ้น

*เมื่อใดจึงควรใช้*แบบจำลอง Spiral

- เมื่อต้องนำส่งโครงการบ่อยๆ
- เมื่อโครงการมีขนาดใหญ่
- เมื่อความต้องการไม่ชัดเจนและซับซ้อน
- เมื่อโครงการอาจมีการเปลี่ยนแปลงได้ตลอดเวลา
- โครงการขนาดใหญ่และงบประมาณสูง

ข้อดี แบบจำลอง Spiral

- มีการวิเคราะห์ความเสี่ยงและลดผลกระทบจากความเสี่ยงสูง
- มีประโยชน์สำหรับโครงการขนาดใหญ่และสำคัญต่อภารกิจ

ข้อเสีย แบบจำลอง Spiral

- เป็นโมเดลที่มีต้นทุนสูงได้
- การวิเคราะห์ความเสี่ยงจำเป็นต้องมีความชำนาญเป็นพิเศษ
- ใช้งานไม่ได้กับโครงการขนาดเล็ก

แบบจำลอง V

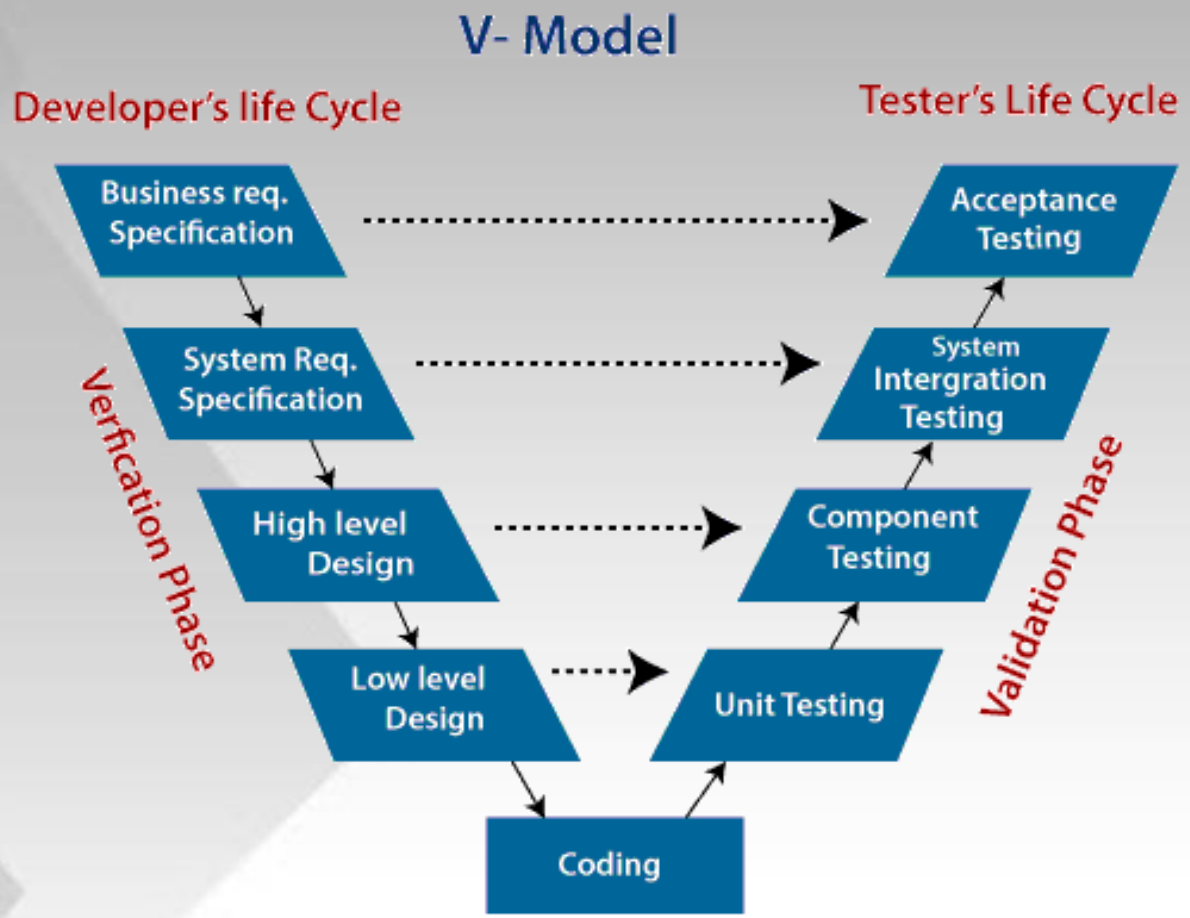
(V Model)

แบบจำลอง V (*V-Model)*

ในการทดสอบและการพัฒนาแบบจำลอง SDLC ประเภทนี้ มีการวางแผนขั้นตอนควบคู่กันไป ดังนั้นจึงมีขั้นตอนการตรวจสอบที่ด้านข้างและขั้นตอนการตรวจสอบความถูกต้องในอีกด้านหนึ่ง V-Model เข้าร่วมโดยขั้นตอนการเข้ารหัส

V-Model เรียกอีกอย่างว่า Verification and Validation Model ในการนี้ แต่ละเฟสของ SDLC จะต้องทำให้เสร็จก่อนเริ่มเฟสถัดไป เป็นไปตามกระบวนการออกแบบตามลำดับ เช่นเดียวกับแบบจำลองน้ำตก การทดสอบอุปกรณ์มีการวางแผนควบคู่ไปกับขั้นตอนการพัฒนาที่สอดคล้องกัน

แบบจำลอง V (*V-Model) (ต่อ)*



แหล่งที่มา https://www.javatpoint.com/

แบบจำลอง V (*V-Model) (ต่อ)*

การยืนยัน-Verification: มันเกี่ยวข้องกับวิธีการวิเคราะห์แบบคงที่ (ตรวจสอบ) ที่ทำโดยไม่ต้องรันโค้ด เป็นกระบวนการประเมินกระบวนการพัฒนาผลิตภัณฑ์เพื่อค้นหาว่าตรงตามข้อกำหนดที่ระบุหรือไม่

การตรวจสอบความถูกต้อง-Validation: มันเกี่ยวข้องกับวิธีการวิเคราะห์แบบไดนามิก (การทำงาน, ไม่ทำงาน) การทดสอบทำได้โดยการรันโค้ด การตรวจสอบความถูกต้องเป็นกระบวนการในการจำแนกซอฟต์แวร์หลังจากเสร็จสิ้นกระบวนการพัฒนาเพื่อพิจารณาว่าซอฟต์แวร์นั้นตรงตามความคาดหวังและความต้องการของลูกค้าหรือไม่

ดังนั้น V-Model จึงมีขั้นตอนการตรวจสอบความถูกต้องที่ด้านหนึ่งของขั้นตอนการตรวจสอบความถูกต้องในอีกด้านหนึ่ง กระบวนการตรวจสอบและยืนยันจะเข้าร่วมโดยขั้นตอนการเข้ารหัสในรูปตัววี ดังนั้นจึงเรียกว่า V-Model

ขั้นตอนต่างๆ ของ *Verification Phase* ของ *V-model*

1. การวิเคราะห์ความต้องการทางธุรกิจ-Business requirement analysis: นี่เป็นขั้นตอนแรกที่เข้าใจความต้องการของผลิตภัณฑ์จากฝั่งลูกค้า ระยะนี้มีการสื่อสารโดยละเอียดเพื่อทำความเข้าใจความคาดหวังของลูกค้าและความต้องการที่แน่นอน

2. การออกแบบระบบ-System Design: ในขั้นตอนนี้ วิศวกรระบบจะวิเคราะห์และตีความธุรกิจของระบบที่เสนอโดยศึกษาเอกสารข้อกำหนดของผู้ใช้

3. การออกแบบสถาปัตยกรรม-Architecture Design: พื้นฐานในการเลือกสถาปัตยกรรมคือควรเข้าใจทั้งหมดซึ่งโดยทั่วไปประกอบด้วยรายการของโมดูล ฟังก์ชันการทำงานโดยย่อของแต่ละโมดูล ความสัมพันธ์ของอินเทอร์เฟซ การขึ้นต่อกัน ตารางฐานข้อมูล ไดอะแกรมสถาปัตยกรรม รายละเอียดเทคโนโลยี ฯลฯ การทดสอบการรวม แบบจำลองจะดำเนินการในขั้นตอนเฉพาะ

ขั้นตอนต่างๆ ของ *Verification Phase* ของ *V-model*(ต่อ)

4. การออกแบบโมดูล-Module Design: ในขั้นตอนการออกแบบโมดูล ระบบจะแบ่งออกเป็นโมดูลขนาดเล็ก มีการระบุการออกแบบโดยละเอียดของโมดูล ซึ่งเรียกว่าการออกแบบระดับต่ำ

5. ขั้นตอนการเข้ารหัส: หลังจากออกแบบ ขั้นตอนการเข้ารหัสจะเริ่มต้นขึ้น ขึ้นอยู่กับข้อกำหนด ภาษาโปรแกรมที่เหมาะสมจะถูกตัดสินใจ มีแนวทางและมาตรฐานบางประการสำหรับการเข้ารหัส ก่อนตรวจสอบในที่เก็บ บิลด์สุดท้ายได้รับการปรับให้เหมาะสมเพื่อประสิทธิภาพที่ดีขึ้น และโค้ดต้องผ่านการตรวจสอบโค้ดจำนวนมากเพื่อตรวจสอบประสิทธิภาพ

ขั้นตอนต่างๆ ของ *Validation Phase* ของ *V-model*

การทดสอบเป็นหน่วยย่อย-Unit Testing: ในแบบจำลอง V แผนการทดสอบหน่วย (UTP) ได้รับการพัฒนาในระหว่างขั้นตอนการออกแบบโมดูล UTP เหล่านี้ถูกดำเนินการเพื่อขจัดข้อผิดพลาดที่ระดับโค้ดหรือระดับหน่วย หน่วยเป็นเอนทิตีที่เล็กที่สุดซึ่งสามารถดำรงอยู่ได้โดยอิสระ เช่น โมดูลโปรแกรม การทดสอบหน่วยตรวจสอบว่าเอนทิตีที่เล็กที่สุดสามารถทำงานได้อย่างถูกต้องเมื่อแยกจากโค้ด/หน่วยที่เหลือ

การทดสอบการทำงานร่วมกัน-Integration Testing: แผนการทดสอบการบูรณาการได้รับการพัฒนาในระหว่างขั้นตอนการออกแบบสถาปัตยกรรม การทดสอบเหล่านี้ยืนยันว่ากลุ่มที่สร้างและทดสอบอย่างอิสระสามารถอยู่ร่วมกันและสื่อสารกันเองได้

ขั้นตอนต่างๆ ของ *Validation Phase* ของ *V-model (ต่อ)*

การทดสอบระบบ-System Testing: แผนการทดสอบระบบได้รับการพัฒนาในช่วงการออกแบบระบบ ซึ่งแตกต่างจากแผนการทดสอบหน่วยและการรวมระบบ แผนการทดสอบระบบประกอบด้วยทีมธุรกิจของลูกค้า การทดสอบระบบช่วยให้มั่นใจว่าเป็นไปตามความคาดหวังจากนักพัฒนาแอปพลิเคชัน

การทดสอบการยอมรับ-Acceptance Testing: การทดสอบการยอมรับเกี่ยวข้องกับส่วนการวิเคราะห์ความต้องการทางธุรกิจ รวมถึงการทดสอบผลิตภัณฑ์ซอฟต์แวร์ในบรรยากาศของผู้ใช้ การทดสอบการยอมรับเผยให้เห็นปัญหาความเข้ากันได้กับระบบต่างๆ ที่มีอยู่ในบรรยากาศของผู้ใช้ ร่วมกันค้นพบปัญหาที่ไม่ทำงานเช่นข้อบกพร่องด้านโหลดและประสิทธิภาพภายในบรรยากาศของผู้ใช้จริง

ควรใช้ V-Model เมื่อใด

- เมื่อความต้องการมีการกำหนดไว้อย่างดีและไม่คลุมเครือ
- ควรใช้แบบจำลองรูปตัววีสำหรับโครงการขนาดเล็กถึงขนาดกลางที่มีการกำหนดและกำหนดข้อกำหนดไว้อย่างชัดเจน
- ควรเลือกแบบจำลองรูปตัววีเมื่อมีแหล่งข้อมูลทางเทคนิคตัวอย่างพร้อมผู้เชี่ยวชาญทางเทคนิคที่จำเป็น

ข้อได้เปรียบ (ข้อดี) ของ V-Model:

- เข้าใจง่าย
- วิธีการทดสอบ เช่น การวางแผน การออกแบบการทดสอบ เกิดขึ้นได้ก่อนจะเขียนโค้ด
- ซึ่งช่วยประหยัดเวลาได้มาก จึงมีโอกาสสำเร็จสูงกว่าแบบจำลองน้ำตก
- หลีกเลี่ยงการไหลลงของข้อบกพร่อง
- ทำงานได้ดีสำหรับแผนขนาดเล็กที่เข้าใจข้อกำหนดได้ง่าย

ข้อเสีย (ข้อเสีย) ของ V-Model:

- แข็งมากและยืดหยุ่นน้อยมาก
- ไม่ดีสำหรับโครงการที่ซับซ้อน
- ซอฟต์แวร์ได้รับการพัฒนาในระหว่างขั้นตอนการใช้งาน ดังนั้นจึงไม่มีการผลิตซอฟต์แวร์ต้นแบบในช่วงต้น
- หากมีการเปลี่ยนแปลงระหว่างทาง จะต้องอัปเดตเอกสารการทดสอบพร้อมกับเอกสารที่จำเป็น

แบบจำลอง Incremental

(Incremental Model)

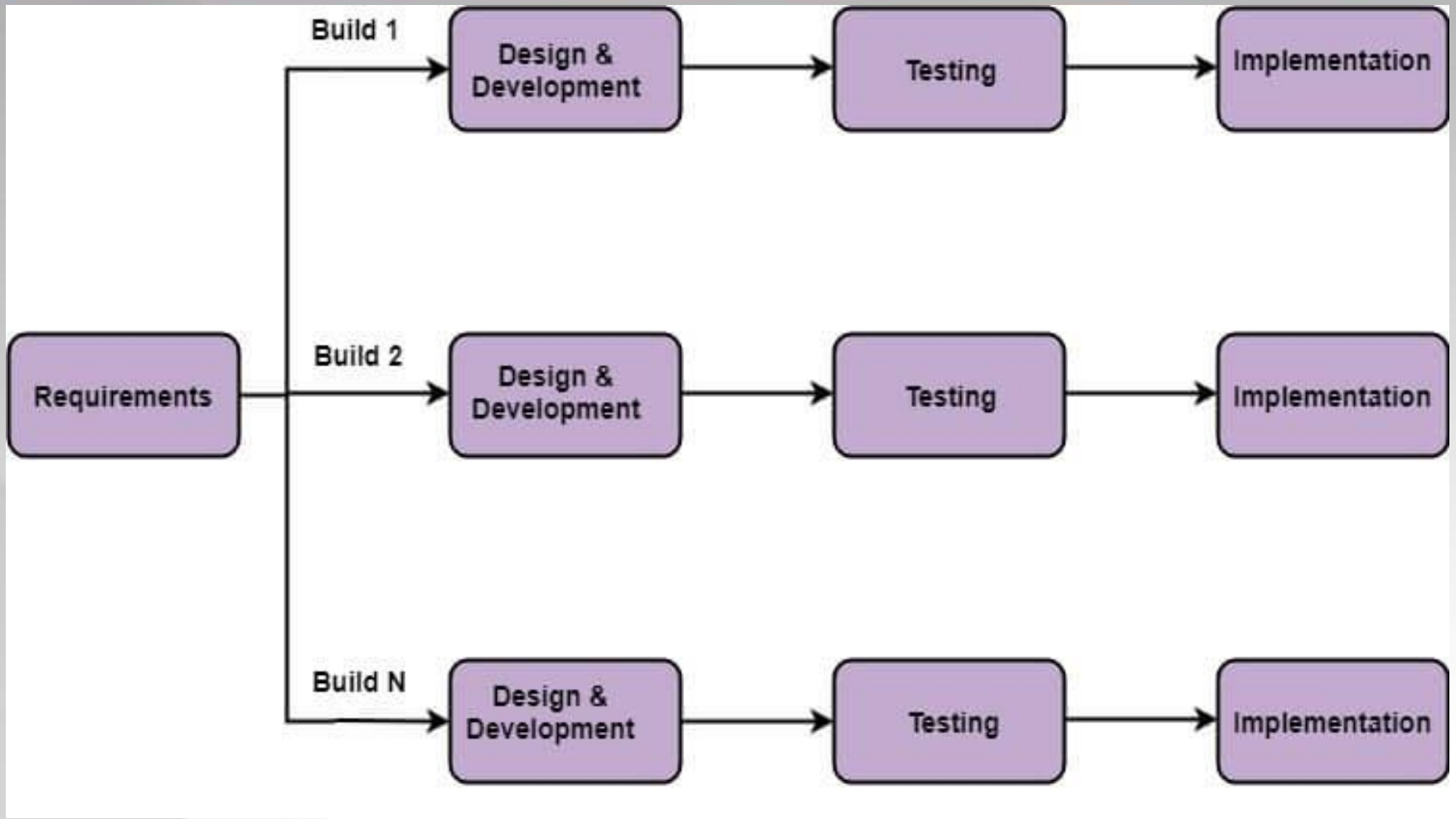
แบบจำลอง Incremental

แบบจำลอง Incremental เป็นโมเดลที่เพิ่มขึ้นไม่ใช่แบบจำลองที่แยกจากกัน มันจำเป็นต้องเป็นชุดของแบบจำลองน้ำตก ข้อกำหนดจะแบ่งออกเป็นกลุ่มต่างๆ เมื่อเริ่มโครงการ สำหรับแต่ละกลุ่มจะติดตามโมเดล SDLC เพื่อพัฒนาซอฟต์แวร์ กระบวนการ SDLC ซ้ำแล้วซ้ำอีก โดยแต่ละรุ่นเพิ่มฟังก์ชันการทำงานมากขึ้นจนกว่าจะตรงตามข้อกำหนดทั้งหมด ในวิธีนี้ แต่ละรอบจะทำหน้าที่เป็นขั้นตอนการบำรุงรักษาสำหรับซอฟต์แวร์รุ่นก่อนหน้า การปรับเปลี่ยนแบบจำลองส่วนเพิ่มช่วยให้รอบการพัฒนาซ้อนทับกันได้ หลังจากนั้นรอบถัดไปอาจเริ่มต้นก่อนที่รอบก่อนหน้าจะเสร็จสมบูรณ์

แบบจำลอง Incremental(ต่อ)

แบบจำลอง Incremental เป็นกระบวนการของการพัฒนาซอฟต์แวร์ที่ความต้องการแบ่งออกเป็นโมดูลแบบสแตนด์อโลนหลายโมดูลของวงจรการพัฒนาซอฟต์แวร์ ในรูปแบบนี้ แต่ละโมดูลจะผ่านข้อกำหนด การออกแบบ การใช้งาน และขั้นตอนการทดสอบ โมดูลรุ่นต่อๆ มาทุกรุ่นจะเพิ่มฟังก์ชันให้กับรุ่นก่อนหน้า กระบวนการจะดำเนินต่อไปจนกว่าระบบจะเสร็จสมบูรณ์

แบบจำลอง Incremental(ต่อ)



แหล่งที่มา https://www.javatpoint.com/

*ขั้นตอนต่างๆ ของ*แบบจำลอง Incremental

1. การวิเคราะห์ความต้องการ-Requirement analysis: ในระยะแรกของแบบจำลองส่วนเพิ่ม ความเชี่ยวชาญในการวิเคราะห์ผลิตภัณฑ์จะระบุข้อกำหนด และข้อกำหนดการทำงานของระบบนั้นเข้าใจโดยทีมวิเคราะห์ความต้องการ ในการพัฒนาซอฟต์แวร์ภายใต้แบบจำลองส่วนเพิ่ม ขั้นตอนนี้มีบทบาทสำคัญอย่างยิ่ง

2. การออกแบบและการพัฒนา-Design & Development: ในระยะนี้ของโมเดลส่วนเพิ่มของ SDLC การออกแบบฟังก์ชันการทำงานของระบบและวิธีการพัฒนาเสร็จสิ้นลงด้วยความสำเร็จ เมื่อซอฟต์แวร์พัฒนาการใช้งานได้จริง รูปแบบที่เพิ่มขึ้นจะใช้รูปแบบและขั้นตอนการพัฒนา

3. การทดสอบ-Testing: ในแบบจำลองส่วนเพิ่ม ขั้นตอนการทดสอบจะตรวจสอบประสิทธิภาพของแต่ละฟังก์ชันที่มีอยู่ตลอดจนฟังก์ชันเพิ่มเติม ในขั้นตอนการทดสอบจะใช้วิธีการต่างๆ เพื่อทดสอบพฤติกรรมของแต่ละงาน

*ขั้นตอนต่างๆ ของ*แบบจำลอง Incremental (ต่อ)

3. การทดสอบ-Testing: ในแบบจำลองส่วนเพิ่ม ขั้นตอนการทดสอบจะตรวจสอบประสิทธิภาพของแต่ละฟังก์ชันที่มีอยู่ตลอดจนฟังก์ชันเพิ่มเติม ในขั้นตอนการทดสอบจะใช้วิธีการต่างๆ เพื่อทดสอบพฤติกรรมของแต่ละงาน

4. การนำไปใช้งาน-Implementation: ระยะการนำไปปฏิบัติช่วยให้ขั้นตอนการเข้ารหัสของระบบการพัฒนา มันเกี่ยวข้องกับการเข้ารหัสขั้นสุดท้ายที่ออกแบบในขั้นตอนการออกแบบและพัฒนาและทดสอบการทำงานในขั้นตอนการทดสอบ หลังจากเสร็จสิ้นระยะนี้ จำนวนการทำงานของผลิตภัณฑ์จะเพิ่มขึ้นและอัปเกรดเป็นผลิตภัณฑ์ระบบขั้นสุดท้าย

ควรใช้ แบบจำลอง Incremental เมื่อใด?

- เมื่อความต้องการที่สิ่งสำคัญที่สุด
- โครงการมีกำหนดการพัฒนาที่ยาวนาน
- เมื่อทีมงานซอฟต์แวร์ไม่ชำนาญหรือผ่านการฝึกอบรมมาเป็นอย่างดี
- เมื่อลูกค้าต้องการปล่อยสินค้าอย่างรวดเร็ว
- คุณสามารถพัฒนาข้อกำหนดที่มีลำดับความสำคัญก่อนได้

*ข้อได้เปรียบของ*แบบจำลอง Incremental

- ข้อผิดพลาดนั้นง่ายต่อการรับรู้
- ง่ายต่อการทดสอบและแก้ไขข้อบกพร่อง
- คล่องตัวมากขึ้น
- ง่ายต่อการจัดการความเสี่ยงเพราะจัดการได้ในระหว่างการทำซ้ำ
- ลูกค้าได้รับฟังก์ชันการทำงานที่สำคัญตั้งแต่เนิ่นๆ

*ข้อเสียของ*แบบจำลอง Incremental

- ต้องมีการวางแผนที่ดี
- ต้นทุนรวมสูง
- จำเป็นต้องมีอินเทอร์เฟซโมดูลที่กำหนดไว้อย่างดี

แบบจำลอง Agile

(Agile Model)

แบบจำลอง Agile

แบบจำลอง Agile เป็นแนวทางปฏิบัติที่ส่งเสริมการทำงานร่วมกันอย่างต่อเนื่องของการพัฒนาและการทดสอบในระหว่างกระบวนการ SDLC ของโครงการใดๆ ในวิธี Agile โปรเจ็กต์ทั้งหมดจะถูกแบ่งออกเป็นบิลด์ที่เพิ่มขึ้นทีละน้อย บิลด์ทั้งหมดนี้มีให้ในการทำซ้ำ และการวนซ้ำแต่ละครั้งจะใช้เวลาตั้งแต่หนึ่งถึงสามสัปดาห์

ระยะซอฟต์แวร์ที่คล่องตัวนั้นมีลักษณะเฉพาะในลักษณะที่กล่าวถึงสมมติฐานหลักหลายประการเกี่ยวกับโครงการซอฟต์แวร์จำนวนมาก

1. เป็นการยากที่จะคิดล่วงหน้าว่าข้อกำหนดของซอฟต์แวร์ใดจะคงอยู่และสิ่งใดที่จะเปลี่ยนแปลง เป็นการยากที่จะคาดการณ์ว่าลำดับความสำคัญของผู้ใช้จะเปลี่ยนไปอย่างไรเมื่อโครงการดำเนินไป

แบบจำลอง Agile(ต่อ)

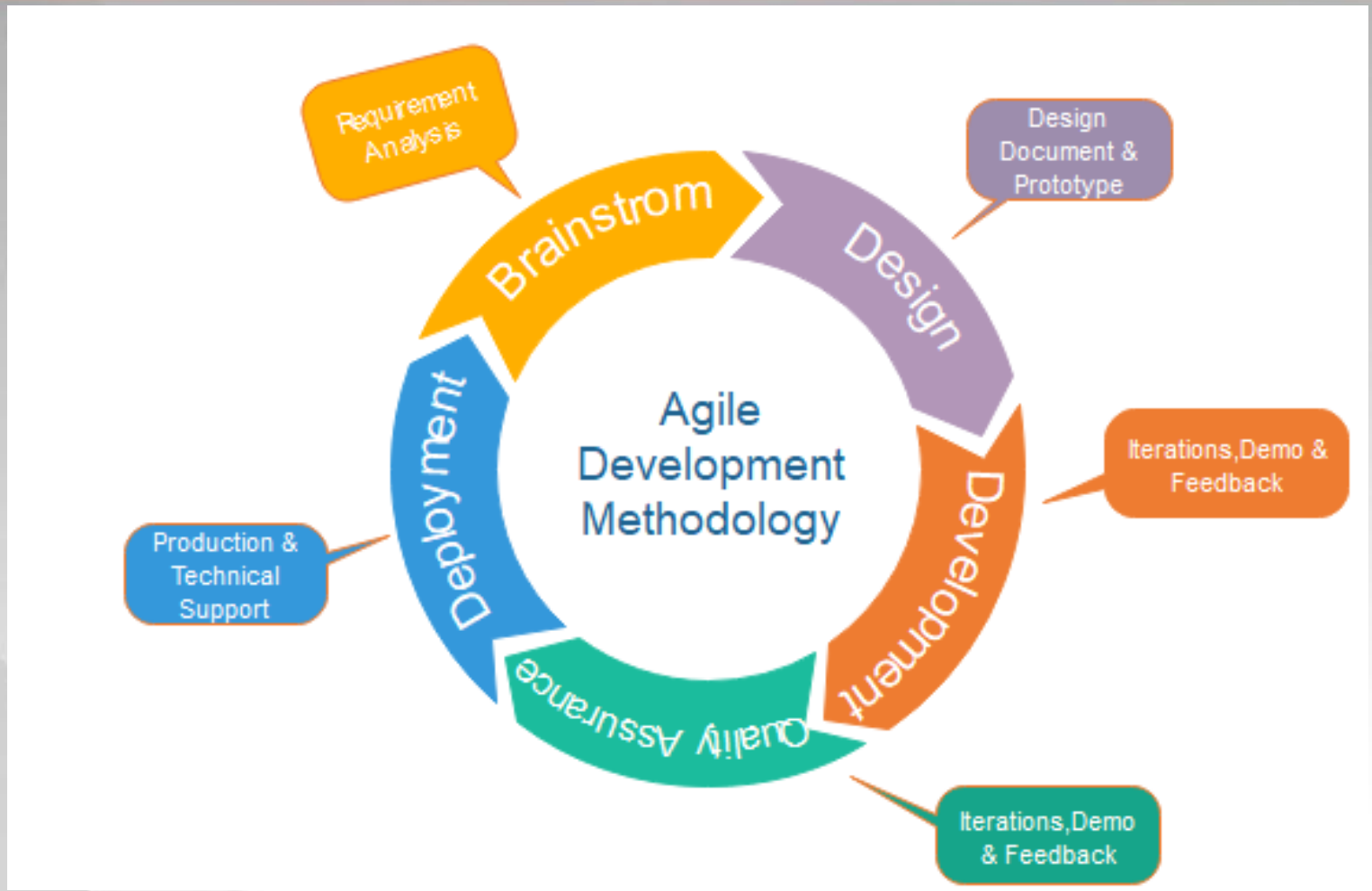
2. สำหรับซอฟต์แวร์หลายประเภท การออกแบบและการพัฒนามีการแทรกซ้อน กล่าวคือ กิจกรรมทั้งสองควรดำเนินการควบคู่กันไปเพื่อให้แบบจำลองการออกแบบได้รับการพิสูจน์ในขณะที่สร้างขึ้น เป็นการยากที่จะคิดว่าการออกแบบมีความจำเป็นมากเพียงใดก่อนที่จะใช้การก่อสร้างเพื่อทดสอบการกำหนดค่า

3. การวิเคราะห์ การออกแบบ การพัฒนา และการทดสอบไม่สามารถคาดเดาได้ (จากมุมมองของการวางแผน) อย่างที่เราต้องการ

แบบจำลอง Agile(ต่อ)

รูปแบบกระบวนการ Agile" หมายถึงแนวทางการพัฒนาซอฟต์แวร์ตามการพัฒนาซ้ำ วิธีการแบบ Agile แบ่งงานออกเป็นการวนซ้ำที่เล็กลง หรือบางส่วนไม่เกี่ยวข้องกับการวางแผนระยะยาวโดยตรง ขอบเขตและข้อกำหนดของโครงการกำหนดไว้ที่จุดเริ่มต้นของกระบวนการพัฒนา แผนเกี่ยวกับจำนวนการทำซ้ำ ระยะเวลา และขอบเขตของการทำซ้ำแต่ละครั้งมีการกำหนดไว้ล่วงหน้าอย่างชัดเจนการทำซ้ำแต่ละครั้งถือเป็น "กรอบ" ในช่วงเวลาสั้นๆ ในแบบจำลองกระบวนการ Agile ซึ่งโดยทั่วไปจะใช้เวลาตั้งแต่หนึ่งถึงสี่สัปดาห์ การแบ่งส่วนโครงการทั้งหมดออกเป็นส่วนเล็กๆ ช่วยลดความเสี่ยงของโครงการและลดข้อกำหนดด้านเวลาการส่งมอบโครงการโดยรวม การทำซ้ำแต่ละครั้งเกี่ยวข้องกับทีมที่ทำงานผ่านวงจรชีวิตการพัฒนาซอฟต์แวร์เต็มรูปแบบ รวมถึงการวางแผน การวิเคราะห์ความต้องการ การออกแบบ การเขียนโค้ด และการทดสอบ ก่อนที่ผลิตภัณฑ์ที่ใช้งานได้จะแสดงให้ลูกค้าเห็น

แบบจำลอง Agile(ต่อ)



แหล่งที่มา https://www.javatpoint.com/

ขั้นตอนของแบบจำลอง Agile

ขั้นตอนต่างๆ แบบจำลอง Agile มีดังนี้

- การรวบรวมความต้องการ-Requirements gathering
- การออกแบบข้อกำหนด-Design the requirements
- การพัฒนา / การวนซ้ำ-Construction/ iteration
- การทดสอบ/การประกันคุณภาพ-Testing/ Quality assurance
- การติดตั้งหรือการนำไปใช้-Deployment
- ข้อเสนอแนะ-Feedback

ขั้นตอนของแบบจำลอง Agile (ต่อ)

1. การรวบรวมความต้องการ: ในขั้นตอนนี้ คุณต้องกำหนดข้อกำหนด คุณควรอธิบายโอกาสทางธุรกิจและวางแผนเวลาและความพยายามที่จำเป็นในการสร้างโครงการ จากข้อมูลนี้ คุณสามารถประเมินความเป็นไปได้ทางเทคนิคและเศรษฐกิจ

2. ออกแบบข้อกำหนด: เมื่อคุณได้ระบุโครงการแล้ว ให้ทำงานร่วมกับผู้มีส่วนได้ส่วนเสียเพื่อกำหนดข้อกำหนด คุณสามารถใช้ Dataflow Diagram หรือ UML Diagram เพื่อแสดงการทำงานของฟีเจอร์ใหม่และแสดงให้เห็นว่าจะนำไปใช้กับระบบที่มีอยู่ของคุณอย่างไร

3. การพัฒนา/ ทำซ้ำ: เมื่อทีมกำหนดข้อกำหนดแล้ว งานก็เริ่มขึ้น นักออกแบบและนักพัฒนาเริ่มทำงานในโปรเจ็กต์ซึ่งมีเป้าหมายในการปรับใช้ผลิตภัณฑ์ที่ใช้งานได้ ผลิตภัณฑ์จะได้รับการปรับปรุงในขั้นตอนต่างๆ ดังนั้นจึงมีฟังก์ชันการทำงานที่เรียบง่ายและน้อยที่สุด

ขั้นตอนของแบบจำลอง Agile (ต่อ)

4. การทดสอบ: ในขั้นตอนนี้ ทีมประกันคุณภาพจะตรวจสอบประสิทธิภาพของผลิตภัณฑ์และค้นหาจุดบกพร่อง

5. การปรับใช้: ในขั้นตอนนี้ ทีมงานจะออกผลิตภัณฑ์สำหรับสภาพแวดล้อมการทำงานของผู้ใช้

6. คำติชม: หลังจากปล่อยผลิตภัณฑ์ ขั้นตอนสุดท้ายคือการตอบรับ ในส่วนนี้ ทีมงานจะได้รับคำติชมเกี่ยวกับผลิตภัณฑ์และดำเนินการตามข้อเสนอแนะ

Agile Testing Methods

- Scrum
- eXtreme Programming(XP)
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development

Scrum

SCRUM เป็นกระบวนการพัฒนาที่คล่องตัวซึ่งมุ่งเน้นไปที่วิธีจัดการงานในเงื่อนไขการพัฒนาแบบทีมเป็นหลัก

มีสามบทบาทในนั้นและความรับผิดชอบคือ:

- Scrum Master: scrum สามารถตั้งมาสเตอร์ทีม จัดประชุม และขจัดอุปสรรคของกระบวนการ
- เจ้าของผลิตภัณฑ์: เจ้าของผลิตภัณฑ์สร้างงานในมือ จัดลำดับความสำคัญของความล่าช้า และรับผิดชอบในการกระจายฟังก์ชันการทำงานในการทำซ้ำแต่ละครั้ง
- Scrum Team: ทีมจัดการงานและจัดระเบียบงานเพื่อให้การวิ่งหรือวงจรเสร็จสมบูรณ์

eXtreme Programming(XP)

วิธีการประเภทนี้จะใช้เมื่อลูกค้ามีการเปลี่ยนแปลงความต้องการหรือข้อกำหนดอยู่ตลอดเวลา หรือเมื่อพวกเขาไม่แน่ใจเกี่ยวกับประสิทธิภาพของระบบ

Crystal

มีสามแนวคิดของวิธีนี้ -

- การเหมารับงาน: มีกิจกรรมรวมอยู่ในขั้นตอนนี้ เช่น การสร้างทีมพัฒนา การวิเคราะห์ความเป็นไปได้ การวางแผนพัฒนา ฯลฯ
- การส่งแบบวนรอบ: ภายใต้วิธีนี้ ประกอบด้วยสองวงรอบคือ

  ทีมงานปรับปรุงแผนการปล่อยผลิตภัณฑ์หรือซอฟต์แวร์

  รวมผลิตภัณฑ์หรือซอฟต์แวร์ทั้งหมดส่งมอบให้ผู้ใช้
- จบงาน: ตามสภาพแวดล้อมของผู้ใช้ เฟสนี้จะดำเนินการติดตั้งและบำรุงรักษาหลังติดตั้ง

Dynamic Software Development Method(DSDM):

วิธีการพัฒนาซอฟต์แวร์แบบไดนามิก (DSDM): DSDM เป็นกลยุทธ์การพัฒนาแอปพลิเคชันอย่างรวดเร็วสำหรับการพัฒนาซอฟต์แวร์และให้โครงสร้างการกระจายโครงการที่คล่องตัว คุณสมบัติที่สำคัญของ DSDM คือผู้ใช้ต้องเชื่อมต่อกันอย่างแข็งขัน และทีมได้รับสิทธิ์ในการตัดสินใจ เทคนิคที่ใช้ใน DSDM คือ:

- Time Boxing
- MoSCoW Rules
- Prototyping

Dynamic Software Development Method(DSDM)(ต่อ)

The DSDM project contains seven stages:

- ก่อนโครงการ
- การศึกษาความเป็นไปได้
- ศึกษาเชิงธุรกิจ
- การทำงานซ้ำของแบบจำลองการทำงาน
- ออกแบบและสร้าง Iteration
- การดำเนินการ
- หลังโครงการ

Feature Driven Development(FDD)

วิธีนี้เน้นที่คุณสมบัติ "การออกแบบและสร้าง" ตรงกันข้ามกับวิธีการอันชาญฉลาดอื่นๆ FDD อธิบายขั้นตอนเล็กๆ ของงานที่ควรได้รับแยกจากกันในแต่ละฟังก์ชัน

Lean Software Development

วิธีการพัฒนาซอฟต์แวร์แบบลีนเป็นไปตามหลักการ "ผลิตได้ทันเวลา" วิธีการแบบลีนบ่งชี้ถึงความเร็วที่เพิ่มขึ้นของการพัฒนาซอฟต์แวร์และการลดต้นทุน การพัฒนาแบบลีนสามารถสรุปได้เป็นเจ็ดขั้นตอน

การกำจัดของเสีย

ขยายการเรียนรู้

เลื่อนความมุ่งมั่น (ตัดสินใจช้าที่สุด)

จัดส่งก่อนกำหนด

เติมพลังให้ทีม

การสร้างความซื่อสัตย์สุจริต

เพิ่มประสิทธิภาพทั้งหมด

ควรใช้แบบจำลอง Agile เมื่อใด

- เมื่อจำเป็นต้องเปลี่ยนแปลงบ่อย
- เมื่อมีทีมงานที่มีคุณภาพและประสบการณ์สูงพร้อมให้บริการ
- เมื่อลูกค้าพร้อมที่จะประชุมกับทีมซอฟต์แวร์ตลอดเวลา
- เมื่อโครงการมีขนาดเล็ก

ข้อดีของแบบจำลอง Agile

- ส่งงานได้บ่อย
- การสื่อสารแบบตัวต่อตัวกับลูกค้า
- การออกแบบที่มีประสิทธิภาพและตอบสนองความต้องการทางธุรกิจ
- การเปลี่ยนแปลงได้ตลอดเวลาเป็นที่ยอมรับ
- ช่วยลดเวลาในการพัฒนาโดยรวม

ข้อเสียของแบบจำลอง *Agile*

- เนื่องจากการขาดแคลนเอกสารที่เป็นทางการ ทำให้เกิดความสับสนและการตัดสินใจที่สำคัญในขั้นตอนต่างๆ อาจถูกตีความผิดได้ทุกเมื่อโดยสมาชิกในทีมที่แตกต่างกัน
- เนื่องจากขาดเอกสารประกอบที่เหมาะสม เมื่อโครงการเสร็จสิ้นและนักพัฒนาจัดสรรให้กับโครงการอื่น การบำรุงรักษาโครงการที่เสร็จสิ้นแล้วจะกลายเป็นปัญหาได้ยาก

แบบจำลอง Iterative

(Iterative Model)

แบบจำลอง Iterative

เป็นการใช้งานเฉพาะของวงจรชีวิตการพัฒนาซอฟต์แวร์ที่มุ่งเน้นไปที่การเริ่มต้นใช้งานที่เรียบง่าย ซึ่งจากนั้นจะมีความซับซ้อนมากขึ้นและชุดคุณลักษณะที่กว้างขึ้นเรื่อยๆ จนกว่าระบบขั้นสุดท้ายจะเสร็จสมบูรณ์ กล่าวโดยย่อ การพัฒนาซ้ำเป็นวิธีการแบ่งการพัฒนาซอฟต์แวร์ของแอพพลิเคชั่นขนาดใหญ่ออกเป็นชิ้นเล็ก ๆ

คุณสามารถเริ่มต้นด้วยข้อกำหนดของซอฟต์แวร์บางอย่างและพัฒนาซอฟต์แวร์เวอร์ชันแรกได้ หลังจากเวอร์ชันแรก หากจำเป็นต้องเปลี่ยนซอฟต์แวร์ ซอฟต์แวร์เวอร์ชันใหม่จะถูกสร้างขึ้นด้วยการทำซ้ำใหม่ การเปิดตัว Iterative Model ทุกครั้งจะเสร็จสิ้นในช่วงเวลาที่แน่นอนและแน่นอนซึ่งเรียกว่าการวนซ้ำIterative Model อนุญาตให้เข้าถึงช่วงก่อนหน้าซึ่งรูปแบบต่างๆ สร้างขึ้นตามลำดับ ผลลัพธ์สุดท้ายของโครงการได้รับการต่ออายุเมื่อสิ้นสุดกระบวนการ Software Development Life Cycle (SDLC)

แบบจำลอง Iterative (ต่อ)

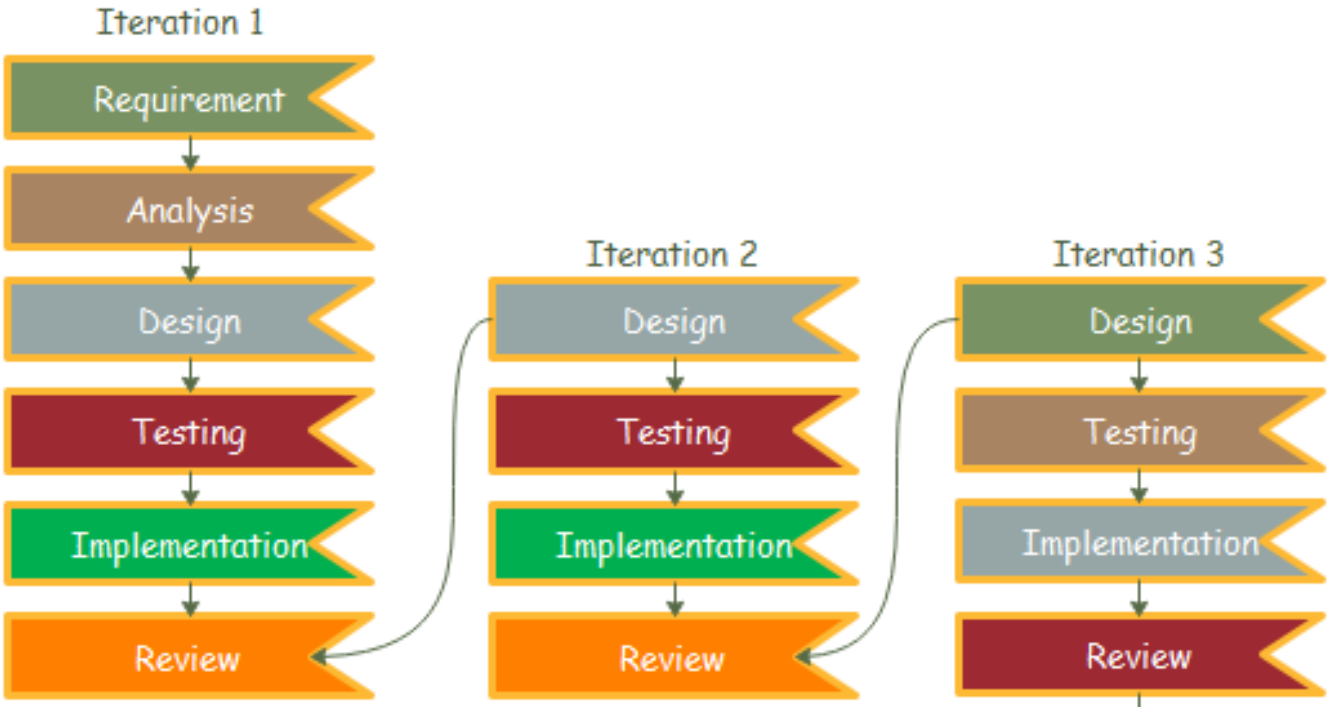


Fig. Iterative Model

แหล่งที่มา https://www.javatpoint.com/

ขั้นตอนต่างๆ ของ *Iterative model*

1. การรวบรวมและวิเคราะห์ความต้องการ: ในระยะนี้ ความต้องการจะถูกรวบรวมจากลูกค้าและตรวจสอบโดยนักวิเคราะห์ว่าข้อกำหนดจะเป็นไปตามข้อกำหนดหรือไม่ นักวิเคราะห์ตรวจสอบความจำเป็นจะบรรลุภายในงบประมาณหรือไม่ หลังจากทั้งหมดนี้ ทีมซอฟต์แวร์จะข้ามไปยังขั้นตอนถัดไป

2. การออกแบบ: ในขั้นตอนการออกแบบ ทีมออกแบบซอฟต์แวร์โดยใช้ไดอะแกรมต่างๆ เช่น ไดอะแกรมการไหลข้อมูล ไดอะแกรมกิจกรรม ไดอะแกรมคลาส ไดอะแกรมการเปลี่ยนสถานะ ฯลฯ

3. Implementation: ในการใช้งาน ข้อกำหนดจะถูกเขียนด้วยภาษาโค้ดและแปลงเป็นโปรแกรมคอมพิวเตอร์ที่เรียกว่าซอฟต์แวร์

ขั้นตอนต่างๆ ของ *Iterative model (ต่อ)*

4. การทดสอบ: หลังจากเสร็จสิ้นขั้นตอนการเข้ารหัส การทดสอบซอฟต์แวร์จะเริ่มโดยใช้วิธีการทดสอบที่แตกต่างกัน มีวิธีการทดสอบหลายวิธี แต่วิธีทดสอบที่พบบ่อยที่สุดคือวิธีทดสอบกล่องขาว กล่องดำ และกล่องสีเทา

5. การปรับใช้: หลังจากเสร็จสิ้นขั้นตอนทั้งหมดแล้ว ซอฟต์แวร์จะถูกปรับใช้กับสภาพแวดล้อมการทำงาน

6. การตรวจทาน: ในขั้นตอนนี้ หลังจากการปรับใช้ผลิตภัณฑ์ ขั้นตอนการตรวจสอบจะดำเนินการเพื่อตรวจสอบลักษณะการทำงานและความถูกต้องของผลิตภัณฑ์ที่พัฒนาแล้ว และหากพบข้อผิดพลาดใดๆ กระบวนการเริ่มต้นอีกครั้งจากการรวบรวมความต้องการ

7. การบำรุงรักษา: ในขั้นตอนการบำรุงรักษา หลังจากการปรับใช้ซอฟต์แวร์ในสภาพแวดล้อมการทำงาน อาจมีจุดบกพร่อง ข้อผิดพลาดบางอย่างหรือจำเป็นต้องมีการอัปเดตใหม่ การบำรุงรักษาเกี่ยวข้องกับการดีบักและตัวเลือกเพิ่มเติมใหม่

ควรใช้แบบจำลอง Iterative เมื่อใด

- เมื่อข้อกำหนดมีการกำหนดไว้อย่างชัดเจนและเข้าใจง่าย
- เมื่อโปรแกรมซอฟต์แวร์มีขนาดใหญ่
- เมื่อมีความต้องการเปลี่ยนแปลงในอนาคต

ข้อดีของ แบบจำลอง Iterative

- การทดสอบและการดีบักในระหว่างการทำซ้ำที่มีขนาดเล็กลงนั้นทำได้ง่าย
- การพัฒนาแบบขนานสามารถวางแผนได้
- เป็นที่ยอมรับได้ง่ายสำหรับความต้องการที่เปลี่ยนแปลงตลอดเวลาของโครงการ
- ความเสี่ยงจะถูกระบุและแก้ไขในระหว่างการทำซ้ำ
- เวลาจำกัดที่ใช้ในการจัดทำเอกสารและมีเวลาเพิ่มเติมในการออกแบบ

ข้อเสียของ แบบจำลอง Iterative

- ไม่เหมาะสำหรับโครงการขนาดเล็ก
- อาจต้องใช้ทรัพยากรเพิ่มเติม
- การออกแบบสามารถเปลี่ยนแปลงได้ครั้งแล้วครั้งเล่าเนื่องจากความต้องการที่ไม่สมบูรณ์
- การเปลี่ยนแปลงความต้องการอาจทำให้เกินงบประมาณ
- วันที่เสร็จสิ้นโครงการไม่ได้รับการยืนยันเนื่องจากข้อกำหนดที่เปลี่ยนแปลง

แบบจำลอง Big Bang

(Big Bang Model)

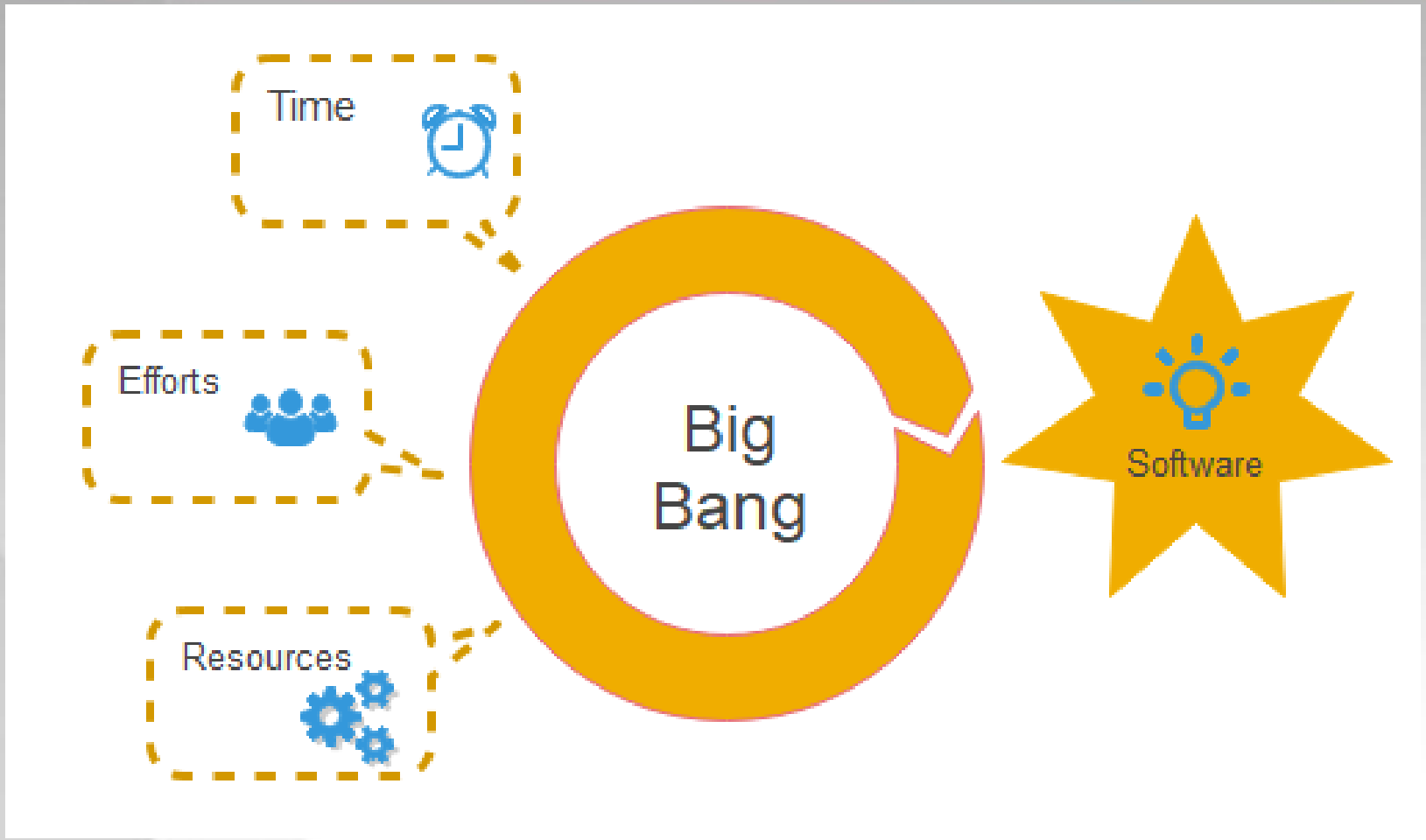
แบบจำลอง Big Bang

โมเดลบิ๊กแบงมุ่งเน้นไปที่ทรัพยากรทุกประเภทในการพัฒนาซอฟต์แวร์และการเข้ารหัส โดยไม่มีการวางแผนหรือมีเพียงเล็กน้อย ข้อกำหนดมีความเข้าใจและดำเนินการ เมื่อมาถึงโมเดลนี้ทำงานได้ดีที่สุดสำหรับโปรเจ็กต์ขนาดเล็กที่มีทีมพัฒนาขนาดเล็กกว่าซึ่งทำงานร่วมกัน นอกจากนี้ยังเป็นประโยชน์สำหรับโครงการพัฒนาซอฟต์แวร์ทางวิชาการอีกด้วย เป็นโมเดลในอุดมคติที่ความต้องการไม่เป็นที่รู้จักหรือไม่ได้ระบุวันที่วางจำหน่ายครั้งสุดท้าย

แบบจำลอง Big Bang(ต่อ)

ในรูปแบบนี้ นักพัฒนาไม่ปฏิบัติตามกระบวนการเฉพาะใดๆ การพัฒนาเริ่มต้นด้วยเงินทุนที่จำเป็นและความพยายามในรูปแบบของปัจจัยการผลิต และผลลัพธ์ที่ได้อาจจะใช่หรือไม่ใช่ตามความต้องการของลูกค้าก็ได้ เพราะในแบบจำลองนี้ แม้แต่ความต้องการของลูกค้าก็ไม่ได้กำหนดไว้โมเดลนี้เหมาะสำหรับโครงการขนาดเล็ก เช่น โครงการวิชาการหรือโครงการภาคปฏิบัติ นักพัฒนาหนึ่งหรือสองคนสามารถทำงานร่วมกันในรูปแบบนี้ได้

แบบจำลอง Big Bang (ต่อ)



แหล่งที่มา https://www.javatpoint.com/

*เมื่อไหร่ควร*แบบจำลอง Big Bang *?*

ดังที่เราได้กล่าวไว้ข้างต้น แบบจำลองนี้จำเป็นเมื่อโครงการนี้มีขนาดเล็ก เช่น โครงการวิชาการหรือโครงการภาคปฏิบัติ วิธีการนี้ยังใช้เมื่อทีมนักพัฒนามีขนาดเล็กและเมื่อไม่ได้กำหนดความต้องการ และลูกค้าไม่ยืนยันหรือกำหนดวันที่วางจำหน่าย

ข้อดีของ แบบจำลอง Big Bang

- ไม่จำเป็นต้องมีการวางแผน
- โมเดลที่เรียบง่าย
- ต้องการทรัพยากรเพียงเล็กน้อย
- ง่ายต่อการจัดการ
- ยืดหยุ่นสำหรับนักพัฒนา

ข้อเสียของ แบบจำลอง Big Bang

- มีความเสี่ยงและความไม่แน่นอนสูง
- ไม่เป็นที่ยอมรับสำหรับโครงการขนาดใหญ่
- หากข้อกำหนดไม่ชัดเจนอาจทำให้มีราคาแพงมาก

แบบจำลอง Prototype

(Prototype Model)

แบบจำลอง Prototype (ต่อ)

แบบจำลองต้นแบบเริ่มต้นด้วยการรวบรวมความต้องการ ผู้พัฒนาและผู้ใช้ตอบสนองและกำหนดวัตถุประสงค์ของซอฟต์แวร์ ระบุความต้องการ ฯลฯ

'การออกแบบอย่างรวดเร็ว' จะถูกสร้างขึ้น การออกแบบนี้มุ่งเน้นไปที่แง่มุมต่างๆ ของซอฟต์แวร์ที่ผู้ใช้จะมองเห็นได้ แล้วนำไปสู่การพัฒนาต้นแบบ จากนั้นลูกค้าจะตรวจสอบต้นแบบ และมีการดัดแปลงหรือเปลี่ยนแปลงใดๆ ที่จำเป็นต่อต้นแบบ

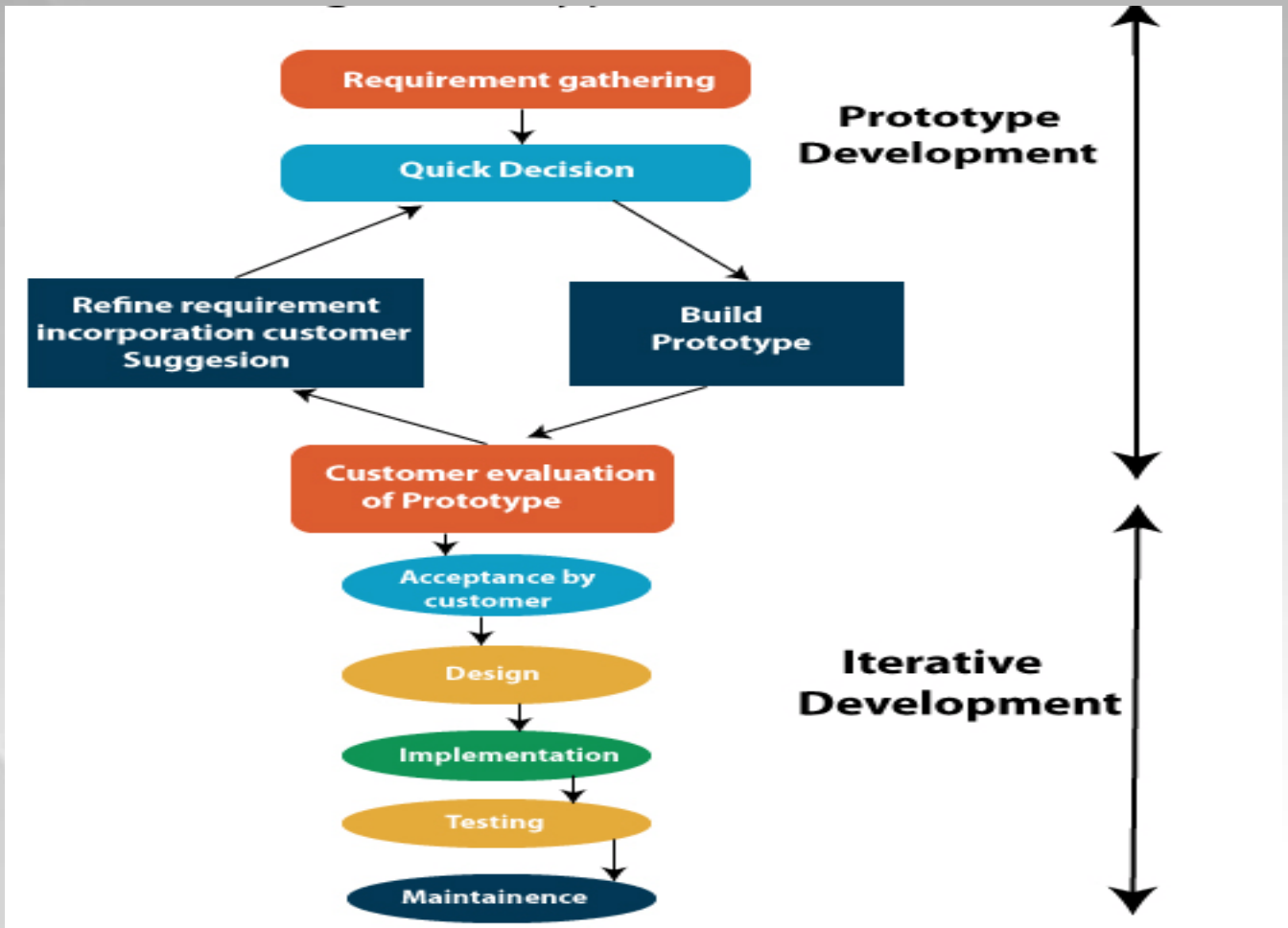
การวนซ้ำเกิดขึ้นในขั้นตอนนี้ และเวอร์ชันต้นแบบที่ดีขึ้นจะถูกสร้างขึ้น สิ่งเหล่านี้จะแสดงให้ผู้ใช้เห็นอย่างต่อเนื่องเพื่อให้สามารถอัปเดตการเปลี่ยนแปลงใหม่ในต้นแบบได้ กระบวนการนี้จะดำเนินต่อไปจนกว่าลูกค้าจะพอใจกับระบบ เมื่อผู้ใช้พอใจแล้ว ต้นแบบจะถูกแปลงเป็นระบบจริงโดยคำนึงถึงคุณภาพและความปลอดภัยทั้งหมด

แบบจำลอง Prototype (ต่อ)

แบบจำลองต้นแบบ กำหนดให้ก่อนดำเนินการพัฒนาซอฟต์แวร์จริง ควรสร้างต้นแบบที่ใช้งานได้ของระบบ ต้นแบบคือการใช้งานระบบของเล่น ต้นแบบมักจะกลายเป็นเวอร์ชันคร่าวๆ ของระบบจริง มีความเป็นไปได้ที่จะแสดงความสามารถในการทำงานที่จำกัด ความน่าเชื่อถือต่ำ และประสิทธิภาพที่ไม่มีประสิทธิภาพเมื่อเทียบกับ

ซอฟต์แวร์จริง ในหลายกรณี ลูกค้ามีมุมมองทั่วไปเกี่ยวกับสิ่งที่คาดหวังจากผลิตภัณฑ์ซอฟต์แวร์เท่านั้น ในสถานการณ์เช่นนี้ที่ไม่มีข้อมูลโดยละเอียดเกี่ยวกับการป้อนข้อมูลเข้าสู่ระบบ ความต้องการในการประมวลผล และข้อกำหนดของผลลัพธ์ อาจใช้แบบจำลองการสร้างต้นแบบ

แบบจำลอง Prototype (ต่อ)



แหล่งที่มา https://www.javatpoint.com/

*ขั้นตอนของ*แบบจำลอง Prototype

- การรวบรวมและวิเคราะห์ความต้องการ
- ตัดสินใจเร็ว
- สร้างต้นแบบ
- การประเมินหรือการประเมินผู้ใช้
- การปรับแต่งต้นแบบ
- ผลิตภัณฑ์

*ข้อดีของ*แบบจำลอง Prototype

- ลดความเสี่ยงจากความต้องการของผู้ใช้ที่ไม่ถูกต้อง
- ดีเมื่อความต้องการมีการเปลี่ยนแปลง/ไม่มีข้อผูกมัด
- กระบวนการช่วยจัดการที่มองเห็นได้เป็นประจำ
- สนับสนุนการตลาดผลิตภัณฑ์ในช่วงต้น
- ลดค่าใช้จ่ายในการบำรุงรักษา
- สามารถตรวจพบข้อผิดพลาดได้เร็วกว่ามาก เนื่องจากระบบทำงานเคียงข้างกัน

*ข้อเสียของ*แบบจำลอง Prototype

- ต้นแบบที่ไม่เสถียร/ใช้งานไม่ดีมักจะกลายเป็นผลิตภัณฑ์ขั้นสุดท้าย
- ต้องการความร่วมมือกับลูกค้าอย่างกว้างขวาง
- เสียเงินลูกค้า
- ต้องการลูกค้าที่มุ่งมั่น
- จบยากถ้าลูกค้าถอนตัว
- อาจจะเจาะจงลูกค้าเกินไป ไม่มีตลาดกว้าง
- ยากที่จะรู้ว่าโครงการจะอยู่ได้นานแค่ไหน

*ข้อเสียของ*แบบจำลอง Prototype

- ง่ายต่อการย้อนกลับไปใช้โค้ดและแก้ไขโดยไม่ต้องมีการวิเคราะห์ การออกแบบ การประเมินลูกค้า และคำติชมที่เหมาะสม
- เครื่องมือสร้างต้นแบบมีราคาแพง
- ต้องใช้เครื่องมือและเทคนิคพิเศษในการสร้างต้นแบบ
- เป็นกระบวนการที่ใช้เวลานาน