

# Software Process: Waterfall vs Agile

อ. ประจักษ์ ปิยะวงศ์วิศาล

Pratch Piyawongwisal

# Today

- Assignment 3
- Recap – Git Commands
- Software Process: Waterfall vs Agile

# Due: Assignment 3

- Create GitHub repo for Team Project
  - Put all documents in: doc/
  - Add Ionic App folder to root directory

# Recap - Git

# Quiz – Git commands

ทบทวน: คำสั่งใดใน **git** ใช้ในการ\_\_\_\_\_?

- สร้าง **checkpoint** บน **local repo**  
(พร้อมกรอก **comment** เกี่ยวกับการแก้ไขโค้ดที่ได้ทำไป)
- ตรวจสอบสถานะของ **repo** ว่ามีไฟล์ใดที่ **git** ยังไม่ได้ติดตามหรือไม่
- เพิ่มไฟล์ให้ **git** ทำการติดตาม (**staging**)
- สร้าง **branch** ใหม่
- กระโดดไปยัง **branch** ใดๆ
- ดาวน์โหลดทั้ง **repo** จากหน้า **GitHub** ที่สนใจนำมาต่อยอด
- ดึงการเปลี่ยนแปลงล่าสุดจาก **remote repo**
- อัปเดตการเปลี่ยนแปลงล่าสุดไปยัง **remote repo**

# Quiz – Git commands

ทบทวน: คำสั่งใดใน **git** ใช้ในการ\_\_\_\_\_?

- สร้าง checkpoint บน local repo (พร้อมกรอก comment เกี่ยวกับการแก้ไขโค้ดที่ได้ทำไป)
- ตรวจสอบสถานะของ repo ว่ามีไฟล์ใดที่ git ยังไม่ได้ติดตามหรือไม่
- เพิ่มไฟล์ให้ git ทำการติดตาม (staging)
- สร้าง branch ใหม่
- กระโดดไปยัง branch ใดๆ
- ดาวน์โหลดทั้ง repo จากหน้า GitHub ที่สนใจนำมาต่อยอด
- ดึงการเปลี่ยนแปลงล่าสุดจาก remote repo
- อัปโหลดการเปลี่ยนแปลงล่าสุดไปยัง remote repo

git commit

git status

git add

git branch

git checkout

git clone

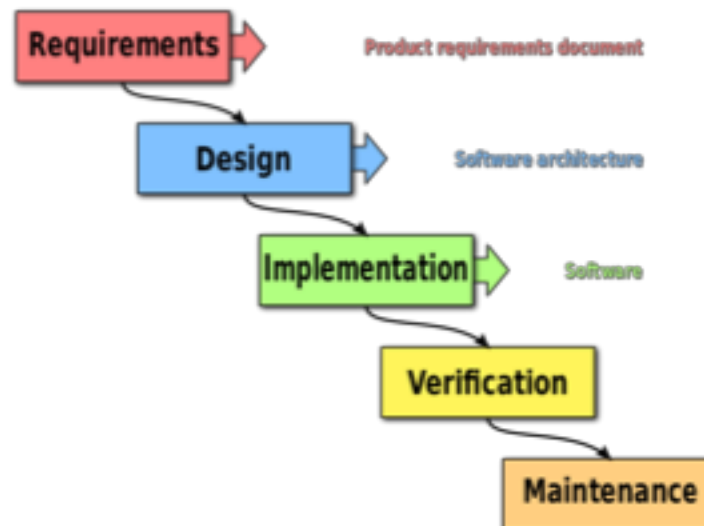
git pull

git push

# Software Process

# Recap: Software Process

- กระบวนการพัฒนาซอฟต์แวร์ ประกอบด้วย กิจกรรมพื้นฐาน 4 อย่างหลักๆ ได้แก่
  1. การจัดทำข้อกำหนดซอฟต์แวร์ (Software Specification)
  2. การออกแบบและผลิตซอฟต์แวร์ (Software Design and Implementation)
  3. การตรวจสอบซอฟต์แวร์ (Software Validation)
  4. การวิวัฒนาการของซอฟต์แวร์ (Software Evolution)





# Software Process

## 1. การจัดทำข้อกำหนดซอฟต์แวร์ (Software Specification)

- ติดต่อสื่อสารกับลูกค้า
- นิยามฟังก์ชันการทำงานของซอฟต์แวร์
- ระบุข้อจำกัดต่าง ๆ เช่น ทางนโยบาย ทรัพยากร เวลา ความปลอดภัย เป็นต้น

\*ข้อกำหนด มักเรียกทับศัพท์ว่า **requirement**

# Software Process

## 2. การออกแบบและผลิตซอฟต์แวร์ (Software Design and Implementation)

- นำข้อกำหนดมาขยายความให้ละเอียดเป็น System Requirement
- เลือกใช้เครื่องมือในการพัฒนา
- ออกแบบระบบทั้งหมดโดยใช้แผนภาพต่างๆ เช่น UML diagrams
- ลงมือพัฒนา ผลิตซอฟต์แวร์ให้ตรงตามข้อกำหนด

# Software Process

## 3. การตรวจสอบซอฟต์แวร์ (Software Validation)

- ตรวจสอบความถูกต้องของโปรแกรมตาม **requirement** ของลูกค้า

- มักแบ่งเป็นหลายกิจกรรม ดังนี้

- Unit testing

ทดสอบส่วนเล็กๆ ในโปรแกรม เช่น คลาส ฟังก์ชัน

- Integration testing

ทดสอบการทำงานร่วมกันของหลายๆ unit

- Acceptance testing

ทดสอบการยอมรับของผู้ใช้/ลูกค้า

- Code review

ให้เพื่อนร่วมทีมอ่านโค้ดเพื่อตรวจสอบคุณภาพของโค้ด

# Software Process

## 4. การวิวัฒนาการของซอฟต์แวร์ (Software Evolution)

- บำรุงรักษาซอฟต์แวร์ (maintenance)
- ปรับปรุงซอฟต์แวร์ตามความต้องการที่เปลี่ยนไปของลูกค้า

# Methodologies / Process Model

ระเบียบวิธีการพัฒนาซอฟต์แวร์ (**Software Development Methodology**) หรือ  
โมเดลกระบวนการพัฒนาซอฟต์แวร์ (**Software Process Model**) เป็นแนวคิดเกี่ยวกับการ  
กำหนดกระบวนการพัฒนาซอฟต์แวร์ให้เป็นลำดับขั้นตอนที่ชัดเจน สามารถนำไปปฏิบัติได้

แต่ละโมเดลอาจมีลักษณะการไหลของกระบวนการ (**Process Flow**) ที่ต่างกัน เช่น

- แบบเชิงเส้น (**Linear**)
- แบบวนซ้ำ (**Iterative**)

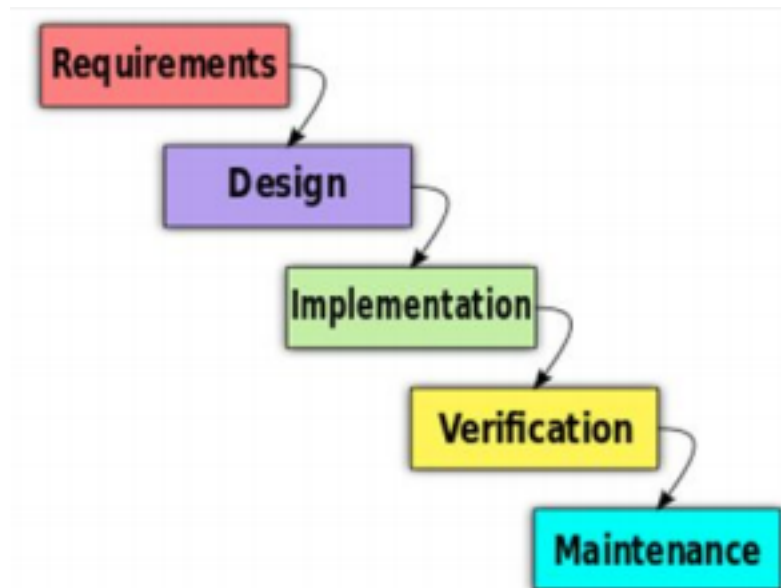
สองโมเดลที่เป็นที่นิยม

- **Waterfall Model**                      => **Linear**
- **Agile Model**                              => **Iterative/Incremental**

# Waterfall Model

กระบวนการไหลแบบ **linear** โปรเจคแบ่งเป็นออกเป็น **phase** ๆ โดยที่การทำงานจะเริ่มจากเฟสแรก ไปจนถึง เฟสสุดท้าย โดยไม่มีการย้อนกลับไปยังเฟสก่อนหน้า

- |               |                                  |
|---------------|----------------------------------|
| • Requirement | เก็บความต้องการของลูกค้า         |
| • Design      | ออกแบบและเลือกใช้เครื่องมือ      |
| • Development | พัฒนาซอฟต์แวร์                   |
| • Implement   | ลงมือเขียนโค้ด                   |
| • Verify      | ตรวจสอบความถูกต้อง               |
| • Deploy      | นำไปใช้งานจริง                   |
| • Maintenance | รันระบบ เฝ้าสังเกต และแก้ไขปัญหา |



# Waterfall Model

- วิธีการนี้เป็นที่นิยมในช่วงปี **1970-80** (เกิดจากอุตสาหกรรมก่อสร้าง/การผลิต)
- มีลักษณะทำงานเป็นเส้นตรง (**Linear**)
- แต่ละเฟสจะไม่ทับซ้อนกัน -- ต้องทำเฟสปัจจุบันให้เสร็จก่อนจึงจะไปเฟสต่อไปได้
- ย้อนไปเฟสก่อนหน้าไม่ได้
- ใช้เวลานานกับเฟสแรกๆ (**Requirement, Design**)

ปัจจุบันผู้คนใน **industry** เห็นตรงกันว่า **waterfall** เป็น **model** ที่ไม่ดี

- แต่ก็อาจยังเหมาะกับบางงาน

# Waterfall Model

## ข้อดี

- ใช้เวลาในช่วง Requirement/Design มาก ทำให้เจอ bug ตั้งแต่เนิ่นๆ
  - ยิ่งเจอ bug ซ้ำไปในเฟสหลังๆ ยิ่งเกิดความเสียหาย ค่าใช้จ่ายจะยิ่งมาก
- ได้ทำ documentation ตั้งแต่เฟส Requirement/Design
  - ความรู้จาก doc เป็นประโยชน์ต่อนักพัฒนาทุกคน
- มีการวางแผนตั้งแต่ต้นจนจบโดยละเอียด ทำให้ทีมงานเข้าใจ problem space ได้อย่างถ่องแท้
- กระบวนการทั้งหมดเข้าใจง่าย



# Waterfall Model

## ข้อเสีย

- ไม่เหมาะกับ **requirement** ที่เปลี่ยนแปลงได้
  - ลูกค้าเปลี่ยนใจ
- ยากที่จะตัดสินใจวางแผนทั้งหมดตั้งแต่แรก เพราะยังไม่เขียนโค้ดจริง จะไม่เห็นปัญหา
- **stakeholder** ไม่ได้เข้ามามีส่วนร่วมเท่าไรนัก
- ขาด **feedback** จาก **stakeholder** ที่ใช้ในการปรับปรุงส่วนต่างๆ

# Iterative/Incremental Model

สร้าง **product** ทีละนิดๆ เป็น **iteration** มีการส่งมอบงานทุก **iteration** แต่ละ **iteration** ประกอบด้วย **process** เหล่านี้

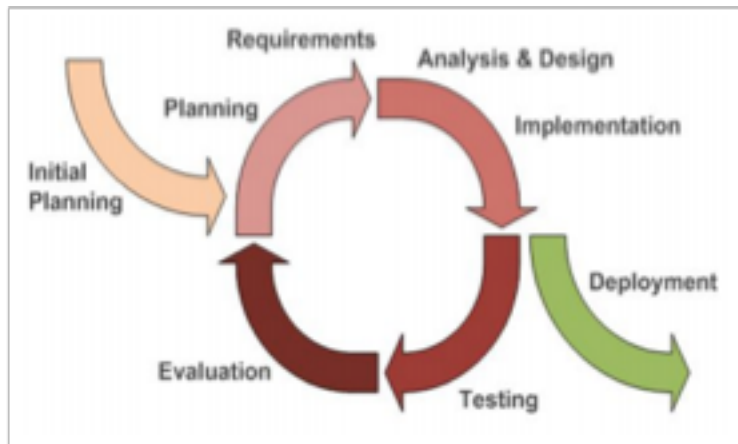
- **Planning/Requirement**
- **Design/Implementation**
- **Testing**
- **Evaluation**

วางแผน/เก็บความต้องการของลูกค้า

ออกแบบและพัฒนา

ตรวจสอบความถูกต้อง

ประเมินคุณภาพจากการใช้งานจริง



# Iterative/Incremental Model

## ข้อดี

- ทำงานเป็น **iteration** สั้นๆ = มีโอกาสที่จะรับ **feedback** จากผู้ใช้แล้วค่อยๆ ปรับปรุง
- **release** งานหลังจบ **iteration** ทำให้ผู้ใช้ได้มีส่วนร่วมในการพัฒนามากขึ้น
- ลดค่าใช้จ่ายในการปรับแก้หากลูกค้าเปลี่ยนใจ

# Iterative/Incremental Model

## ข้อเสีย

- ขาดการวางแผนในระยะยาว/ในภาพรวมทั้งหมด อาจทำให้เกิดปัญหาในระดับสถาปัตยกรรม ซึ่งแก้ไขยาก
- ไม่เหมาะกับอุตสาหกรรมบางประเภท
  - เช่น ผู้ผลิต **Microprocessor** อย่าง **Intel** คงยังนิยมใช้ **Waterfall** มากกว่า

กระนั้น ยิ่งเวลาผ่านไป ทีมพัฒนาซอฟต์แวร์ก็ต่างมีแนวโน้มที่จะไปในทาง **iterative** มากขึ้นเรื่อยๆ  
เมื่อถึงจุดหนึ่ง จึงเกิดคำว่า **"Agile"**

# Agile Manifesto

ในปี **2001** นักพัฒนาซอฟต์แวร์ **17** คนได้มาหารือกันที่รีสอร์ทแห่งหนึ่งในเมือง **Snowbird, Utah** อเมริกา หลังจากที่ได้ถกกันในเรื่องแนวการพัฒนาซอฟต์แวร์แบบใหม่แล้ว ได้ข้อสรุปตรงกัน จึงได้ตีพิมพ์เอกสาร **Agile Manifesto** ออกมาเผยแพร่



**Manifesto** = คำแถลงอุดมการณ์

# Agile Manifesto

เนื้อหาความในคำแถลงนี้โดยคร่าว กล่าวถึง **2** สิ่ง

- **Software Process**
- **Team culture**

มีประเด็นเด่นๆ คือ

- ส่งเสริมการสร้างวัฒนธรรมการทำงานเป็นทีม
- **focus** ไปที่ปลายทาง นั่นคือตัวซอฟต์แวร์ที่จะส่งให้ลูกค้าเป็นสำคัญ
- ต้องเก็บ **feedback** จากลูกค้า
- พร้อมที่จะ **adapt** ต่อการเปลี่ยนแปลงเสมอ



ทีมนักพัฒนาสมัยใหม่ต่างเห็นตรงกันว่าวิธีการนี้**เหมาะ**กับการพัฒนาซอฟต์แวร์จริง คือใช้แล้ว **work**

# Agile Process

จากหลักการ **Agile** นำไปสู่ **process** ที่นำไปใช้ได้จริง -> เหล่านี้คือ **Agile Process** ที่เป็นที่ยอมรับ

- XP
- Scrum
- Kanban

ในคาบถัดๆ ไป เราจะศึกษาวิธีการแบบ **Scrum** โดยละเอียด

Ionic



# Ionic – News App (cont.)

- นำข้อมูลข่าวจาก **newsapi.org** มาแสดงผลในรูปแบบ **card**
- ดัดแปลงให้ **card** แสดงแค่ **headline** แล้วลิงค์ไปหน้า **single** ที่มีเนื้อความข่าว

# Ionic – News App (สรุป)

- สร้างหน้าใหม่ใน Ionic ใช้คำสั่ง
  - `ionic g page <name>`
- เมื่อสร้างหน้าใหม่แล้ว สามารถปรับการ **route url** ได้ ในไฟล์ **app.routing**
- แต่ละ **page** จะประกอบด้วย 2 ส่วนที่สำคัญ ได้แก่
  - **.html** ใช้วาง **component <ion-xyz>** ต่างๆ (เรียกว่าส่วน **View** ใน MVC)
  - **.ts** ใช้กำหนดพฤติกรรมของโปรแกรม **logic** ต่างๆ (เรียกว่าส่วน **Controller** ใน MVC)
- **service** เหมาะกับ **background task** ที่ต้องการให้หลายหน้าเรียกใช้ได้ เช่น โหลดไฟล์ ติดต่อเซิร์ฟเวอร์
  - สร้างได้โดยใช้คำสั่ง `ionic g service <name>`
- การส่ง **http request** ไปตามเว็บไซต์ ให้ใช้ **HttpClient**
  - ก่อนใช้ ต้องนำเข้า **HttpClientModule** จาก `@angular/core/http`
  - สร้าง **ionic service** ที่ **implement** ฟังก์ชัน **getData()** ภายในมีการเรียกใช้ **http.get()**
  - **page.ts** เรียกใช้ **service** อีกที แล้วนำผลลัพธ์ส่งไปแสดงผลที่ **.html**

# Intermediate Git - Branching

- อ้างอิง slide #4 - Git