20 เรื่องที่นักพัฒนา Software ควรรู้

somkiat.cc

สมเกียรติ ปุยสูงเนิน

เรื่องที่ 1

Coding with reason

โค้ดทุกๆ ตัวอักษร ควรเกิดมาอย่างมีเหตุผล

ในการเขียน code นั้นควรมีเหตุผลในการเขียนเสมอ

ไม่ใช่เขียนตามใจฉัน

โดยเหตุผลในการเขียน code ประกอบไปด้วย

- มีความถูกต้อง
- code มันสื่อถึงความเข้าใจในปัญหาที่คุณแก้ไข
- code มันคือสิ่งที่ต้องสื่อสารออกไปให้คนอื่นเข้าใจ และได้ประโยชน์จากมัน ไม่ใช่รับกรรมหรือโยนขี้

แต่แปลกนะ ในระบบงานต่างๆ

เรามักพบว่ามันจะมี code ที่มัน error

หรือทำงานผิดพลาด มากกว่า code ที่ทำงานได้อย่างถูกต้อง

เนื่องจากเราทำการตรวจสอบความถูกต้องของ code กัน

แบบแปลกๆ หรือ ช้าไปไหม หรือ ไม่มีประสิทธิภาพ ?

ดังนั้น สิ่งที่เราต้องการก็คือ เครื่องมือทำงานแบบอัตโนมัติเข้ามาช่วย แต่ไม่จำเป็นเสมอไปนะ

Code ควรที่จะแบ่งออกเป็นกลุ่มย่อยๆ

ในแต่ละ method/function หรือ แต่ละ block ไม่ควรมีจำนวน Line of Code เยอะนะ

ตัวอย่างเช่นไม่น่าเกิน 10 บรรทัด

ถ้ามีจำนวน Line of Code เยอะเกินไป ก็จะใช้เวลาการตรวจสอบความถูกต้องเยอะเช่นกัน

ดังนั้นลดจำนวน Line of Code ลงน่าจะเป็นทางที่ดีกว่า

และในแต่ละส่วนของ code ควรที่จะมีหน้าที่การทำงานอย่างใดอย่างหนึ่งไปเลย (Single Responsibility Principle)

ดังนั้น ถ้ามีส่วนการทำงานใดของ code มีหน้าที่การทำงานมากกว่าหนึ่งอย่าง ให้ทำการแยกส่วนการทำงานออกมาซะ

ส่งผลให้การตรวจสอบความถูกต้อง และ ความเข้าใจของ code เร็วและดีขึ้นอย่างอัตโนมัตินั่นเอง

มีรูปแบบในการเขียน code ที่ดีแนะนำ

ในปัจจุบันมีรูปแบบการเขียน code ที่ดีมากมาย ซึ่งทุกๆ รูปแบบนั้นสามารถตรวจสอบด้วย Static Code Analyzer ได้ทั้งหมด ดังนี้

- หลีกเลี่ยงการใช้คำสั่ง goto ผมเชื่อว่าคงไม่มีใครใช้แล้ว
- หลีกเลี่ยงการใช้งาน Global variable ที่สามารถแก้ไขให้ใช้งานจากที่ไหนๆ ก็ได้ ควรใช้งานได้เฉพาะในส่วนที่เกี่ยวข้องเท่านั้น
- แต่ละตัวแปร ควรมีขอเขตการใช้งานที่แคบที่สุด
- ทำให้ code อ่านง่ายๆ ด้วยการใช้ spacebar ที่เหมาะสมและเหมือนกันทั้งหมด
- code ควรที่จะอธิบายตัวมันเอง ดังนั้นเรื่องของการตั้งชื่อจึงมีความสำคัญมากๆ
- ถ้าในส่วนการทำงานหนึ่งๆ มีการทำงานที่ซับซ้อน หรือหลายขั้นตอน ในขั้นแรกให้ทำการแยกออกมาเป็น method/function ใหม่ก่อนเลย

- แต่ละ method/function ควรที่จะสั้นๆ และทำงานเพียงอย่างเดียว นั่นคือให้ทำการกำหนดจำนวน Line of Code ไปเลย ถ้าเป็นแต่ก่อนจะบอกว่าไม่เกิน 24 บรรทัด หรือ หนึ่งหน้าจอ แต่ในปัจจุบัน 10 บรรทัดก็มากเกินไปแล้ว
- แต่ละ method/function ควรมีจำนวน parameter ให้น้อยที่สุด ดีที่สุดคือไม่มี parameter เลย หรือมากที่สุดไม่ควรเกิน 4 เนื่องจากยิ่งมีจำนวน parameter มากก็จะเพิ่มหน้าที่และการทำงานสูง รวมทั้งการตรวจสอบความถูกต้องก็ยากและใช้เวลาสูงขึ้น
- code แต่ละส่วนไม่ควรผูกมัดกัน ควรจะติดต่อสื่อสารกันผ่าน abstraction layer หรือ interface แทน

ทั้งหมดนี้

คือเหตุผลที่ developer ควรเขียน code มิใช่เขียนเพียงให้มันเสร็จๆ ไปเท่านั้นนะครับ มันต้องมีเหตุผลเสมอ

เรื่องที่ 2

Code Review

หันกลับมาทบทวน code ที่เขียนกันสักหน่อย

คำถาม

ทำไมเราถึงต้องทำล่ะ ?

คำตอบ

เพราะว่าเราต้องการเพิ่มคุณภาพของ code และลดจำนวน defect/bug ลงไงล่ะ

แต่เหตุผลคงไม่เพียงพอ เพราะว่านักพัฒนาส่วนใหญ่ก็ไม่ทำกัน !!

ความเข้าใจผิด

ผมเชื่อว่าหลายคนคงมีประสบการณ์ที่ไม่ดีนักสำหรับการทำ code review และนักพัฒนาบางคนอาจจะไม่ชอบการทำ code review เลย

เนื่องจากหลายองค์กร หรือ ทีมพัฒนาทำ code review แบบเป็นทางการมากๆ เช่นมีแผนก หรือ หน่วยงาน หรือ กลุ่มคน ที่ทำการ review code

เสมือนว่าเป็น gate of code quality หรือ audit เพื่อคอยจับผิด ก่อนที่จะทำการ deploy code ไปยัง production server

มักจะพบว่า คนที่ทำหน้าที่ review code จะเป็นกลุ่มคนในตำแหน่ง Architect หรือ Tech Lead
การ review code จะมีเพียงมุมมองเดียวตามคนที่ทำการ review
การ review code มักจะทำหลังจากการพัฒนาเสร็จสิ้น
มักจะใช้เวลาในการ review code สูงมาก
และงานในส่วนนี้มักจะกลายเป็นปัญหาคอขวดอีกต่างหาก

สิ่งที่ควรทำความเข้าใจใหม่

การ review code นั้นไม่ใช่ เพื่อค้นหาสิ่งที่ไม่ถูกต้อง
แต่เป้าหมายที่แท้จริงก็คือ *การแบ่งปันความรู้ของคนภายในทีมพัฒนา และช่วยกันสร้างแนวทางในการ coding ของทีม หรือกลายเป็นมาตรฐานที่ทุกๆ คนช่วยกันสร้าง*

ในการแบ่งปันความรู้กันนั้น ทำให้แต่ละคนลดความเป็นเจ้าของ code ในแต่ละส่วนลงไป เนื่องจาก code มันคือ code ของทีม และในแต่ละครั้งควรสุ่มคนในทีมขึ้นมาทำการ review code ให้ทำการอ่านและดู code เพื่อที่จะเรียนรู้และ ทำความเข้าใจมากกว่าตรวจหาข้อผิดพลาด

ในการ review code ควรใช้คำพูดเพื่อวิจารณ์ในเชิงสร้างสรรค์ มากกว่าการพูดดูถูก ถากถาง ไม่เช่นนั้นกิจกรรมนี้มันจะไม่สนุก และ ไร้ประโยชน์ไปในที่สุด

รวมทั้งให้ระวังด้วยสำหรับสมาชิกในทีมที่มีหลากหลายระดับหรือตำแหน่ง ไม่เช่นนั้นกลายเป็นว่าพวก senior เป็นคนนำพาไป หรือ เป็นกลุ่มคนที่ review อยู่ฝ่ายเดียว ซึ่งก็ทำให้กิจกรรมนี้ไร้ซึ่งความหมาย หรือ ไม่เป็นไปตามเป้าหมายที่แท้จริงไปเลย

แต่ละคนในทีมจะมีมุมมองในการ review ที่หลากหลาย ซึ่งทำให้การ review เกิดประโยชน์สูงสุด

- บางคนมองในเรื่องของเอกสาร
- บางคนมองในเรื่องของการจัดการ exception ต่างๆ
- บางคนมองในเรื่องของแต่ละ feature

คำถาม

ช่วงเวลาใดเหมาะสมสำหรับการทำ code review ?

คำตอบ

โดยปกติจะทำอย่างน้อยสัปดาห์ละ 1 ครั้ง ครั้งละ 1-2 ชั่วโมง สำหรับคน review ให้สุ่ม หรือ หมุนเวียนไปในทุกๆ ครั้ง และในแต่ละคนให้ลองเปลี่ยนมุอมอง หรือ role การ review ไปในทุกๆ ครั้ง

ยิ่งคนที่มีประสบการณ์น้อยๆ หรือเพิ่งจบมา ยิ่งต้องเข้าร่วม code review ด้วย เนื่องจากจะได้เห็นการ review ในมุมมองที่แตกต่างออกไป รวมทั้งคนที่มีประสบการณ์ และ ความรู้ก็จำเป็นเช่นเดียวกัน

ถ้าในทีมพัฒนามีการ pair programming กันอยู่แล้ว จะช่วยให้การ review code ลื่นไหลมากขึ้นไปอีก

ในการ review code จะลื่นไหล และ ง่ายมากๆ ถ้าทีมมีเครื่องมือช่วยตรวจสอบเรื่อง coding standard ดังนั้นจะไม่มีการพูดคุย หรือ ถกเถียงเรื่องการจัดรูปแบบของ code กันอีกเลย ส่งผลให้การ review code มีประสิทธิภาพสูงมาก

ขอเน้นย้ำ สิ่งที่สำคัญมาก

Code review ต้องเป็นกิจกรรม หรือ การประชุม ที่มีความสนุกสนาน การ review code นั้น มันคือเรื่องของคนล้วนๆ ดังนั้น ถ้าเป็นการประชุมที่เครียด หรือ ไม่สนุกแล้ว มันก็เป็นการยากที่จะทำให้แต่ละคนทำการ review ได้ดี และมีประสิทธิภาพ

กิจกรรมนี้ไม่ควรมีรูปแบบเป็นทางการ

เนื่องจากเป้าหมายของมันคือ การแบ่งปันความรู้ของคน

ภายในทีม หยุดการพูด comment แบบดูถูก เสียดสี

เปลี่ยนเป็นการพูดคุยแบบสนุกสนาน และ เอาขนมและอาหาร

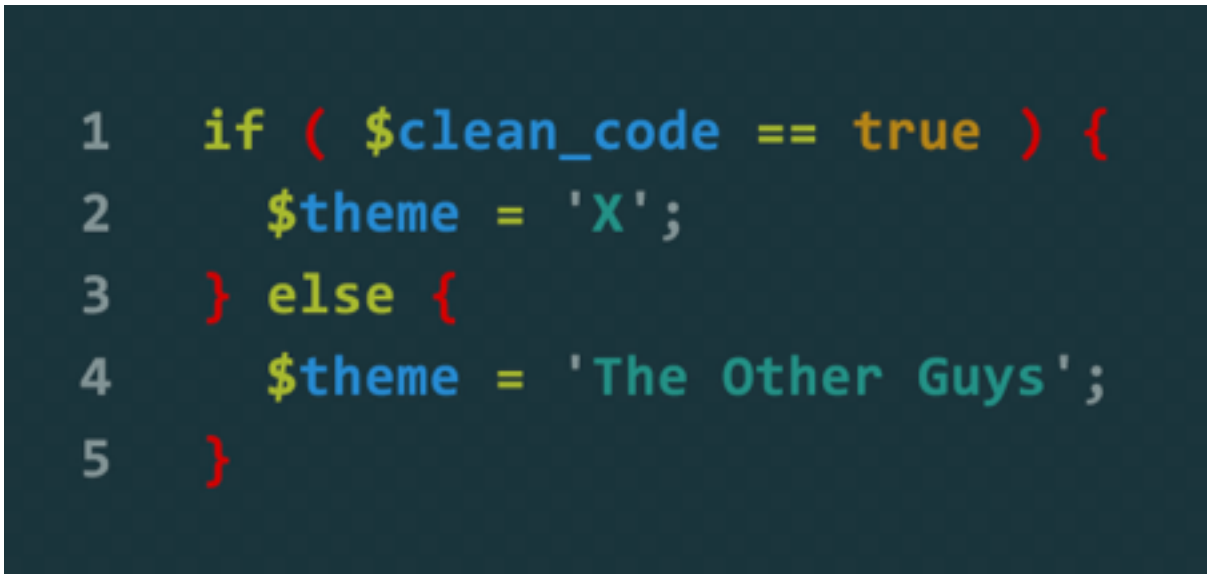
ต่างๆ มานั่งกินแบบสบายๆ ดีกว่า

วันนี้คุณทำ Code review แล้วหรือยัง ?

เรื่องที่ 3

Read Code

มาหัดอ่าน code กันสักนิด



ใครอ่าน code ไม่เป็นบ้าง ?

นักพัฒนา software เป็นสิ่งมีชีวิตที่แปลกประหลาดมากๆ
คือ ชอบที่จะเขียน code ขึ้นมา
แต่แปลกนะ ไม่ชอบอ่าน code ที่เขียนเอง หรือ คนอื่นเขียนขึ้นมา

ดังนั้น การเขียน code นั้นจะสนุกมาก แต่การอ่าน code เป็นเรื่องที่ยากมาก มันแปลกดีนะ !!

บ่อยครั้งนักพัฒนา software มักจะต้องอ่าน code ของคนอื่นอยู่อย่างเสมอ ซึ่งมันยากมากๆ เพราะว่าอะไรก็ตามที่เราไม่ได้เขียนเอง จะอ่านยากไปหมด
เนื่องจากวิธีการแก้ไขปัญหาของแต่ละคนแตกต่างกัน

แต่การอ่าน และ ทำความเข้าใจใน code ของคนอื่น
มันคือการปรับปรุงความสามารถของคุณนะ
เพื่อให้เห็นมุมมอง แนวคิด จากคนอื่นๆ !!!

ดังนั้นในครั้งต่อไป เมื่อต้องอ่าน code

แนะนำให้หยุด และ คิด สักหน่อย

ว่า code ที่เรากำลังอ่านนั้น มันอ่านยาก หรือ ง่าย ?

ถ้ามันอ่านยาก ให้หยุดคิดหน่อยว่า

- มันอ่านยาก เพราะว่าอะไร ?
- มันอ่านยาก เพราะว่ารูปแบบของ code แย่ๆ หรือไม่ ?
- มันอ่านยาก เพราะว่าการตั้งชื่อตัวแปร method และ class ไม่ดีใช่ไหม ?
- มันอ่านยาก เพราะว่ามันรวมปัญหา หรือ ระบบงาน หลายๆ อย่างเข้าด้วยกัน เยอะไปหมด ?
- มันอ่านยาก เพราะว่ารูปแบบของภาษาโปรแกรมที่ใช้พัฒนา ?

ถ้าคุณรู้สาเหตุของปัญหา code ที่อ่านยากแล้ว

คุณจะต้องพยายามเรียนรู้จากข้อผิดพลาดของคนอื่นๆ และของตัวเราเอง เพื่อที่จะไม่สร้างมันขึ้นมาอีก *(แต่แปลกนะ มักจะทำผิดเหมือนๆ กันอยู่อย่างเสมอ)*

หรือในบางครั้งอาจะเจอ code ที่สวยงาม

มีโครงสร้างของ code ที่ดี มีการจัดการ dependency

ต่างๆ อย่างเทพ

code แยกออกเป็น modular หรือผูกมัดกันแบบหลวมๆ

แต่ว่า code เหล่านั้น มันอ่านยากมากๆ !!

ถ้า code มันอ่านง่ายล่ะ

เราควรต้องหยุด และ คิด เช่นเดียวกันว่า

code เหล่านั้นมันดีอย่างไร ?

code เหล่านั้นมันมีอะไรที่เป็นประโยชน์ต่อเรา ?

ให้ทำการเรียนรู้ ทำความเข้าใจกับมัน แล้วนำมาใช้ เช่น

- รูปแบบในการเขียน code ที่สวยงาม เช่น Design pattern ที่เราอาจจะรู้ หรือ ไม่รู้จัก
- การตั้งชื่อที่ดี เป็นอย่างไร
- จำนวนบรรทัดของแต่ละ method/function ที่มันน้อยๆ

แนะนำให้อ่าน code ของ opensource ต่างๆ

เพื่อศึกษาว่า เขาเขียน code กันอย่างไร

ให้มันอ่านง่าย เข้าใจง่าย ถ้าไม่เชื่อก็ลองไป download code จากที่เหล่านี้มาดู

- JUnit
- IntelliJ IDEA

สำหรับการทำงานจริง

คุณในฐานะของนักพัฒนา software

ยังไงก็ต้องทำการอ่าน code เก่าๆ หรือ มักจะถูกเรียกว่า Legacy code

ดังนั้นให้ หยุด คิด มองว่านั่นคือการเรียนรู้

เพื่อเพิ่มทักษะ ประสบการณ์ ให้รู้และเข้าใจมากขึ้นกว่าเดิม

ส่วนการเริ่มต้นนั้น

แนะนำให้คุณลองกลับไปอ่าน code เก่าๆ ที่คุณสร้างขึ้นมาเอง ว่า code มันเป็นอย่างไร ดีหรือไม่ดีอย่างไร ?

จากนั้นทำการแก้ไขปรับปรุงมันให้ดีขึ้น

ทำให้มันอ่านง่ายขึ้นซะ

หรือลองกลับไปดู code ในงานที่คุณกำลังทำอยู่

ไปหา code ที่มันอ่านยากสิ แล้วกลับมาดูว่า คุณกำลังสร้าง code รูปแบบนั้นขึ้นมาหรือไม่ ?

ถ้าใช่ ... ก็ให้ทำการแก้ไข และ ปรับปรุงมันซะ

แล้วจะได้ไม่สร้าง code ให้เป็นภาระต่อคนรุ่นหลัง

ดังนั้น ถ้าครั้งต่อไปคุณรู้สึกว่า

จำเป็นต้องปรับปรุงความสามารถในการ coding แล้ว

อาจจะไม่จำเป็นต้องไปอ่านหนังสืออะไร

แต่ให้ทำการอ่าน code ดีกว่า (Read Code)

ปล. แต่ถ้าอยากอ่านหนังสือ แนะนำให้นักพัฒนาอ่านหนังสือเหล่านี้ครับ หนังสือที่ทีมพัฒนา software ควรที่จะต้องอ่านไว้บ้าง (http://www.somkiat.cc/books-developer-should-read/)

เรื่องที่ 4

Improve code by remove it !

รู้จักลบ code เสียบ้างนะ

อีกคำหนึ่งก็คือ **Less is more** หรือ การทำน้อยแต่ได้มาก

code ก็เช่นเดียวกัน เขียนให้น้อยๆ

แต่สามารถแก้ไขปัญหาได้

รวมทั้งเขียนในสิ่งที่จำเป็นเท่านั้น

คำถาม

ปัจจุบันเราเขียน code อย่างไรกันหนอ ?

ถ้าใครรู้จักกับ Extreme Programming น่าจะเคยได้ยินคำว่า **You Aren't Gonna Need It (YAGNI)**

คือการทำในสิ่งที่เราจำเป็น หรือ ต้องการใช้ในขณะนั้นเท่านั้น

ไม่ต้องไปเผื่ออนาคตอะไรมากนัก !

จำไว้ว่า ถ้าตอนนี้คุณยังไม่จำเป็นต้องใช้งาน ก็ไม่จำเป็นต้องสร้างมันขึ้นมาหรอกนะ

แต่แปลกนะ !!

นักพัฒนาบางคนมักจะเขียน code เผื่อไว้เสมอ

บ่อยครั้งที่มันไม่ถูกใช้งาน มันยังไงกันนะ ?

และที่แปลกมาก คือ code เหล่านั้นมันก็คงอยู่ในระบบของเรา แถมไม่มีใครคิดจะลบ หรือ ลบมันออกไป เพราะว่ากลัวงานเข้า !!

ดังนั้น สิ่งที่นักพัฒนา software ควรจะต้องทำก็คือ

- เขียน code ที่มีความเรียบง่าย (แต่มันไม่ง่ายเลยนะ)
- ปรับปรุงประสิทธิภาพของ code อยู่อย่างสม่ำเสมอ
- ลดจำนวน code ที่มันซ้ำซ้อนในระบบทิ้งไปซะ ... แล้วหา code ส่วนนี้อย่างไรนะ !!
- ควรลบ code ในส่วนที่มันแปลกๆ หรือไม่ใช้งานออกไปบ้าง
- แต่สิ่งที่ควรจะทำไปด้วยคือ การเขียน **unit test** ไม่เช่นนั้นคุณจะรู้ได้อย่างไรว่า code ยังทำงานได้อย่างถูกต้อง หลังจากที่ลบ code บางส่วนออกไป

ความเรียบง่าย และ ความละเอียดในการเขียน code มันสะท้อนถึงประสบการณ์ของนักพัฒนา

คำถามที่น่าสนใจ เกี่ยวกับการเขียน code

- ทำไมเราเขียน code ที่ไม่จำเป็นขึ้นมาตั้งแต่แรกล่ะ ?
- ทำไมเรารู้สึกว่าต้องเขียน code พิเศษๆ ขึ้นมา เพื่อแก้ไขปัญหา ?
- เราทำการ review code และทำ pair programming กันหรือไม่ หรือ ถ้าทำเราทำ เราทำมากันอย่างไรนะ ?

คำตอบที่มักจะได้รับกลับมา

- นักพัฒนาต้องการเขียน code นั้นๆ ขึ้นมา เพื่อเพิ่มคุณค่าให้เกิดขึ้นในระบบ
- นักพัฒนาบางคนบอกว่า code ชุดนี้อาจจะถูกใช้งาน หรือ จำเป็นในอนาคต นั่นคือ การเผื่อไว้ก่อน นั่นคือขัดแย้งกับ YAGNI
- นักพัฒนาบางคนก็ชอบ copy code จากงานเก่าๆ มาใช้ และมักจะบอกว่านี่คือการ reuse !!

- code ชุดพิเศษที่นักพัฒนาเขียนขึ้นมานั้น มันเป็นชุดเล็กๆ ไม่ได้ซับซ้อนอะไรมาก ง่ายๆ นะ แต่เมื่อต้องทำการแก้ไข หรือ ทำการดูแลรักษา มันจะกลายเป็น code ที่ดูแลรักษา และ จัดการยากมากๆ และถ้ามี code พิเศษชุดเล็กๆ เหล่านี้ขึ้นมาเรื่อยๆ ก็ลองคิดดูกันเอาเอง ว่าจะสนุกเพียงใด !
- นักพัฒนาชอบเพิ่มความสามารถต่างๆ ขึ้นมาเอง ตามความรู้สึก ประสบการณ์ โดยที่ทางลูกค้าไม่ได้ต้องการเลย หรือบางครั้งมักจะบอกว่า ความสามารถส่วนนี้เราแถมให้ ซึ่งไม่ค่อยมีคุณค่าต่อผู้ใช้งานหรือลูกค้าเท่าไรนัก

ให้จำไว้ง่ายๆ ว่า

สิ่งที่คุณกำลังทำหรือสร้างอยู่ในตอนนี้คืออะไร ?

สิ่งนั้นคุณจำเป็นต้องการมันในตอนนี้หรือไม่ ?

ถ้าไม่ ... ก็ลบมันทิ้งไปซะ !!!

ดังนั้น จงเรียนรู้ที่จะลบ code ด้วยนะครับ ไม่ใช่แต่เขียนหรือสร้างมันขึ้นมาอย่างเดียว

somkiat.cc