**ข้อ 1**

n = 10

if n<1

puts "Number Worng!"

else

nub = 1

a = ([1])

until nub > n

a << nub \*\* 3

nub = nub + 1

end

end

**ข้อ 2**

fibo = "{"

def prime\_fib (number)

nub1 = 1

until nub1 > number

ff = "f" << nub1

fibonacci = Hash.new{ |h,k| h[k] = k < 2 ? k : h[k-1] + h[k-2] }

fiboanwer = fibonacci[nub1]

fibo << "p#{nub1} -> #{fiboanwer} , "

puts nub1

nub1 = nub1 + 1

end

return fibo

end

puts prime\_fib (50)

**ข้อ 3.**

def build\_tuple\_array(number)

a = (0 .. 1).to\_a

nub1 = 1

until nub1 > number

b = (0 .. number-1).to\_a

a << b

nub1 = nub1 + 1

end

return a

end

**ข้อ 4**

array\_rus8 = Array.new

nn=0

US\_states.each do |us|

array\_rus8 << us

end

puts array\_rus8

**ข้อ 7.**

abbr\_states = [ ["AK", "Alaska"],

["AL", "Alabama"],

["AR", "Arkansas"],

["AS", "American Samoa"],

["AZ", "Arizona"],

["CA", "California"],

["CO", "Colorado"],

["CT", "Connecticut"],

["DC", "District of Columbia"],

["DE", "Delaware"],

["FL", "Florida"],

["GA", "Georgia"],

["GU", "Guam"],

["HI", "Hawaii"],

["IA", "Iowa"],

["ID", "Idaho"],

["IL", "Illinois"],

["IN", "Indiana"],

["KS", "Kansas"],

["KY", "Kentucky"],

["LA", "Louisiana"],

["MA", "Massachusetts"],

["MD", "Maryland"],

["ME", "Maine"],

["MI", "Michigan"],

["MN", "Minnesota"],

["MO", "Missouri"],

["MS", "Mississippi"],

["MT", "Montana"],

["NC", "North Carolina"],

["ND", "North Dakota"],

["NE", "Nebraska"],

["NH", "New Hampshire"],

["NJ", "New Jersey"],

["NM", "New Mexico"],

["NV", "Nevada"],

["NY", "New York"],

["OH", "Ohio"],

["OK", "Oklahoma"],

["OR", "Oregon"],

["PA", "Pennsylvania"],

["PR", "Puerto Rico"],

["RI", "Rhode Island"],

["SC", "South Carolina"],

["SD", "South Dakota"],

["TN", "Tennessee"],

["TX", "Texas"],

["UT", "Utah"],

["VA", "Virginia"],

["VI", "Virgin Islands"],

["VT", "Vermont"],

["WA", "Washington"],

["WI", "Wisconsin"],

["WV", "West Virginia"],

["WY", "Wyoming"] ]

**ข้อ 9**

my\_test= Language.new("jame","5")

my\_test = description.new()

**ข้อ 11.**

ระบบที่จัดเก็บการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์หนึ่งหรือหลายไฟล์เพื่อที่คุณสามารถเรียกเวอร์ชั่นใดเวอร์ชั่นหนึ่งกลับมาดูเมื่อไรก็ได้ ยกตัวอย่างเช่น นักออกแบบกราฟฟิคหรือเว็บดีไซเนอร์และต้องการเก็บทุกเวอร์ชั่นของรูปภาพหรือเลย์เอาต์ (ซึ่งคุณน่าจะอยากเก็บอยู่) การใช้ Version Control System (VCS) เป็นสิ่งที่ชาญฉลาดมาก เพราะมันช่วยให้คุณสามารถย้อนไฟล์บางไฟล์หรือแม้กระทั่งทั้งโปรเจคกลับไปเป็นเวอร์ชั่นเก่าได้ นอกจากนั้นระบบ VCS ยังจะช่วยให้คุณเปรียบเทียบการแก้ไขที่เกิดขึ้นในอดีต ดูว่าใครเป็นคนแก้ไขคนสุดท้ายที่อาจทำให้เกิดปัญหา แก้ไขเมื่อไร ฯลฯ และยังช่วยให้คุณสามารถกู้คืนไฟล์ที่คุณลบหรือทำเสียโดยไม่ตั้งใจได้อย่างง่ายดาย

**ข้อ 12.**

1. ใช้เวลาตอบสนองได้เร็วขึ้น (Quicker response time) เป็นระบบที่สามารถตอบสนองตามคำสั่งของผู้ใช้ได้อย่างรวดเร็ว ระบบแบบรวมศูนย์ (Centralized systems) เป็นระบบที่มีความล่าช้า เมื่อมีการใช้งานหลาย ๆ อย่างพร้อม ๆ กัน จะต้องใช้เวลาในการตอบสนองมากขึ้น

2. ใช้ต้นทุนน้อยกว่า (Lower costs) การใช้ระบบแบบกระจายสามารถลดปัญหาด้านปริมาณของข้อมูลที่จะส่งไปในระยะทางไกล ๆ ได้ ซึ่งถือว่าเป็นการลดต้นทุนในการสื่อสารทางไกล

3. ปรับปรุงความถูกต้องของข้อมูล (Improved data integrity) ผู้ใช้เครือข่ายเฉพาะพื้นที่ (LAN) มักจะรู้จักข้อมูลในพื้นที่ของตนดี และสามารถตรวจสอบข้อผิดพลาดได้รวด

4. ลดต้นทุนตัวประมวลผลหลัก (Reduced host processor costs) จะสามารถเพิ่มอายุการใช้งานของคอมพิวเตอร์ขนาดใหญ่ (Mainframe) เนื่องจากไม่มีภาวะเกินกำลังในเรื่องการปะมวลผลของส่วนกลาง

5. การเพิ่มความน่าเชื่อถือ (Increased reliability) หากคอมพิวเตอร์หลักในระบบการประมวลผลแบบรวมศูนย์ (Centralized processing systems) ล้มเหลวจะเกิดการขัดข้องทั่วทั้งระบบ แต่ในระบบการประมวลผลแบบกระจาย (Distributed processing systems) บางส่วนของระบบสามารถทำหน้าที่เป็นตัวเสริมหรือสนับสนุนในกรณีที่ตัวประมวลผลใดประสบปัญหาหรือล้มเหลวในการทำงานได้

6. การใช้ทรัพยากรร่วมกัน (Resource sharing) ประโยชน์หลักของการพัฒนาเครือข่ายคอมพิวเตอร์ขนาดเล็กในระบบแบบกระจาย คือ ประโยชน์จากการแบ่งสรรทรัพยากรที่มีราคาแพง อุปกรณ์ในการประมวลผลที่มีความเร็วสูง เครื่องพิมพ์เลเซอร์แบบสี จากสถานีงานอื่น

**ข้อ 13.**

1. มีความปลอดภัยข้อมูลสูง (ความน่าเชื่อถือ)

2. การควบคุมจัดการรวมที่ส่วนกลางทั้งหมด สามารถกำหนดความเป็นมาตรฐานเดียวกันได้ เพราะในระบบฐานข้อมูลจะมีกลุ่มบุคคลที่คอยบริหารฐานข้อมูล กำหนดมาตรฐานต่าง ๆ ในการจัดเก็บข้อมูลในลักษณะเดียวกัน

3. .มีการกำหนดนโยบาย และแนวทางปฏิบัติเหมือนกันทั้งองค์การ

4. เกิดการประหยัด เช่น เครื่องมือเครื่องใช้ วัสดุบางอย่างใช้หมุนเวียน สับเปลี่ยนกันได้ หรือเมื่อซื้อวัสดุ ก็ซื้อรวมกันเป็นจำนวนมากไว้ที่ส่วนกลาง ทำให้ได้ราคาต่อหน่วยถูกลง เป็นต้น

**ข้อ 14.**

ทำการ merge พบว่ามีฟ้อง conflict และส่งผลให้ merge ไม่ผ่าน เพราะ Git จะไม่ยอมให้ย้าย branch ถ้าหากยังมี conflict เช่นนี้อยู่ ดังนั้นมาเก็บข้อผิดพลาด โดยเปิดไฟล์ที่ทำการ mergeจะเห็นดังนี้ส่วนที่เกิด conflict จะถูกคั่นด้วยบรรทัดที่ขึ้นต้นด้วยเครื่องหมาย <<<<<<< , ======= , >>>>>>>จัดการลบ 3 บรรทดนั้นทิ้งไปจากนั้น merge เเล้วเช็คคำสั่ง branch --merged ก็ควรจะเห็นครบ

**ข้อ 15.**

Merge บ่อย ๆทุกครั้งเมื่อคุณทำการเปลี่ยนแปลง หรือ commit source code

จะช่วยลดข้อขัดแย้งต่าง ๆ ลงไปอย่างมากถึงจะเกิดข้อขัดแย้ง ก็เป็นเพียงปัญหาเล็ก ๆซึ่งสามารถแก้ไขได้อย่างง่ายดายนักพัฒนาจะไม่ทำงาน หรือ เปลี่ยนแปลง source code ที่เดียวกันอย่างแน่นอนยกเว้นจะทำงานเดียวกัน หรือ ทำงานด้วยกัน

Single Responsibility Principle (SRP)เป็นหนึ่งในแนวทางการออกแบบระบบงานที่ดี คือ SOLIDการออกแบบที่ดีมันช่วยให้ทีมทำงานร่วมกันได้อย่างดียิ่งแต่ละส่วนการทำงานส่วนเล็กๆ

ทำให้คุณภาพของการออกแบบระบบดีรวมทั้งลดความเสี่ยงจาก Merge conflict

Mob programming Communication is a Key

**ข้อ 16.**

Git เป็น Version Control ตัวหนึ่ง ซึ่งเป็นระบบที่มีหน้าที่ในการจัดเก็บการเปลี่ยนแปลงของไฟล์ในโปรเจ็คเรา มีการ backup code ให้เรา สามารถที่จะเรียกดูหรือย้อนกลับไปดูเวอร์ชั่นต่างๆของโปรเจ็คที่ใด เวลาใดก็ได้ หรือแม้แต่ดูว่าไฟล์นั้นๆใครเป็นคนเพิ่มหรือแก้ไข หรือว่าจะดูว่าไฟล์นั้นๆถูกเขียนโดยใครบ้างก็สามารถทำได้ ฉะนั้น Version Control ก็เหมาะอย่างยิ่งสำหรับนักพัฒนาไม่ว่าจะเป็นคนเดียวโดยเฉพาะอย่างยิ่งจะมีประสิทธิภาพมากหากเป็นการพัฒนาเป็นทีม

ส่วน GitHub เว็บที่ให้บริการพื้นที่จัดเก็บโครงการโอเพ่นซอร์สด้วยระบบควบคุมเวอร์ชันแบบ Git โดยมีจุดประสงค์หลักคือ ทำให้การแบ่งปันและพัฒนาโครงการต่างๆด้วยกันเป็นไปได้ง่ายๆ

**ข้อ 17**

เป็น feature ที่ช่วยให้นักพัฒนาสามารถที่จะทำงานได้สะดวกขึ้น ยกตัวอย่างเช่น มีโค๊ดที่ดีอยู่แล้ว แต่อยากจะแก้ไขอะไรก็ตาม ไม่ให้กระทบกับตัวงานหลัก ก็เพียงแค่สร้าง branch ใหม่ขึ้นมา เมื่อแก้ไขหรือทำอะไรเสร็จแล้ว ก็ค่อยเซฟกลับมาที่ master.

**ข้อ 18.**

fast forward merge Merge Branch บน Git นั้น หาก Commit สุดท้ายของ Branch ปลายทาง เป็น Commit เดียวกับจุดที่แยก Branch ออกมา การ Merge จะได้ผลเป็นแบบ Fast-forward

การpush ไปที่ remote repo จึงต้อง Merge แบบ Fast-forward จะเห็นว่า Branch master ไปอยู่ที่ Commit เดียวกับ Branch ปัจจุบัน และสายของ Commit เป็นเส้นตรง

**ข้อ 19.**

git pull ใช้ดึงความเปลี่ยนแปลงจาก remote มายัง local และรวมเข้าด้วยกัน (มีค่าเท่ากับ fetch+merge)

**ข้อ 20.**

เป็นโมเดลการกระจายอำนาจจากส่วนกลาง โดยมี master เป็นแกนหลัก ขนานไปกับ develop อื่นที่มีอยู่เรียกว่า master develop เมื่อพัฒนาเเล้วเเยก develop มีการใส่ feature ใหม่ๆ จนถึงจุดที่มีเสถียรภาพและพร้อมใช้งาน ก็จะรวมกลับมาที่ master ที่เป็นแกนหลัก