

ໂປຣເຈົກທດສອບ Contested Multiplayer Online Game

ໂດຍ

ນາຍອົງລິ້ນ ທັພງູດ

ຮັບສັນກີກາ 633020429-7

ນາຍພາຍຸງ ຮູ່ເຮືອງ ຈັນທວາຮາ

ຮັບສັນກີກາ 633020389-3

ເສັນອ

ອາຈານຍຶ່ງປະຕິບັດ ພສ.ດຣ. ຊືດສູຮາ ສຸມເລື້ກ

รายงานນີ້ເປັນສ່ວນหนີ່ງຂອງການສຶກຫາວິຊາ ຮັ້ງ SC313504
Software Quality Assurance (ການປະກັນຄຸນກາພອື່ແວ້ວ)
ການຮຽນທີ 1 ປຶກສຶກຫາ 2565
ສາຂາວິທະຍາກາຮຄອມພິວເຕອນ ຄະນະວິທະຍາລັຍກາຮຄອມພິວເຕອນ
ມາຮວິທະຍາລັຍຂອນແກ່ນ
(ເດືອນ ຕຸລາຄົມ ພ.ສ. 2565)

การทดสอบ Contested Multiplayer Online Game

ผลการวิเคราะห์ความต้องการ

แบบแผนทดสอบ

วัตถุประสงค์:

- เพื่อตรวจสอบความสมบูรณ์ของระบบและค้นหาจุดบกพร่องของโปรแกรมให้ได้มากที่สุด
- เพื่อนำผลที่วิเคราะห์ได้นำไปปรับปรุงโปรแกรมในอนาคต

สิ่งที่ต้องการทดสอบ:

- ซอฟต์แวร์
- โปรแกรมทดสอบ Static testing

สิ่งที่ไม่ต้องการทดสอบ:

- hardtware

ผู้ทดสอบ:

- นายอภิชัย พัฒนา รหัสนักศึกษา 633020429-7
- นายชาญรุ่งเรือง จันทรารา รหัสนักศึกษา 633020389-3

ระยะเวลาการทดสอบ:

20 กันยายน 2565 ถึง 17 ตุลาคม 2565

วิธีที่ใช้ในการทดสอบ:

Static testing

เทคนิคที่ใช้ในการทดสอบ:

-

เครื่องมือที่ใช้ทดสอบ:

JArchitect,Understand

Goal	Attribute	Metric
Code Quality	Reusability	Inheritance Tree
	Complexity	Cyclomatic Complexity Max Nesting
	Understandability	Percentage of Comment Count Instance variable
	Maintainability	Line of code Lack of Cohesion

1. Understand

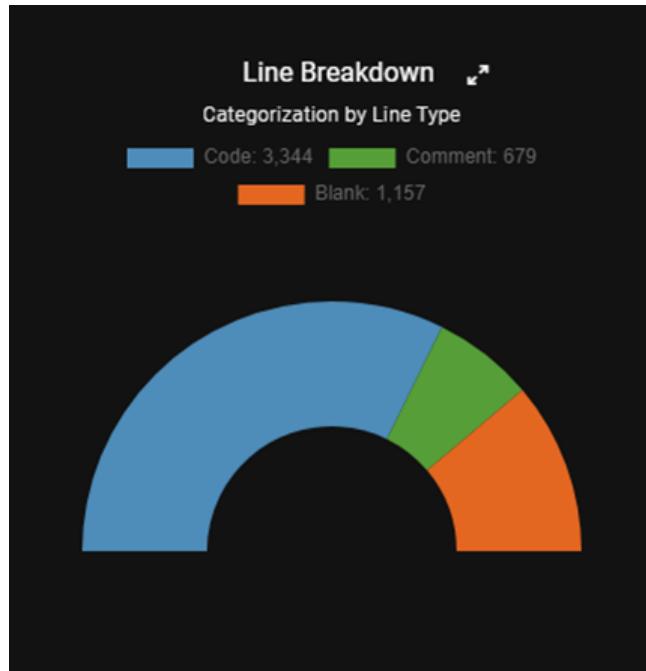
Understand คือ โปรแกรมรองรับการทดสอบแบบ static code ที่จะวิเคราะห์ผลลอกมาในรูปแบบ ภาพ, เอกสาร, หรือเป็นในรูป metric โดยโปรแกรมยังถูกใช้แพร่หลายใน รัฐบาล, การเงิน, วิชาการ และอีกหลากหลายวง

Understand Metric และ overview

1.1 Line Breakdown

แสดงให้เห็นว่าสัดส่วนของโค้ดระหว่าง Code, comment, Blank มีจำนวนบรรทัดเท่าไหร่โดยมีสัดส่วนดังนี้

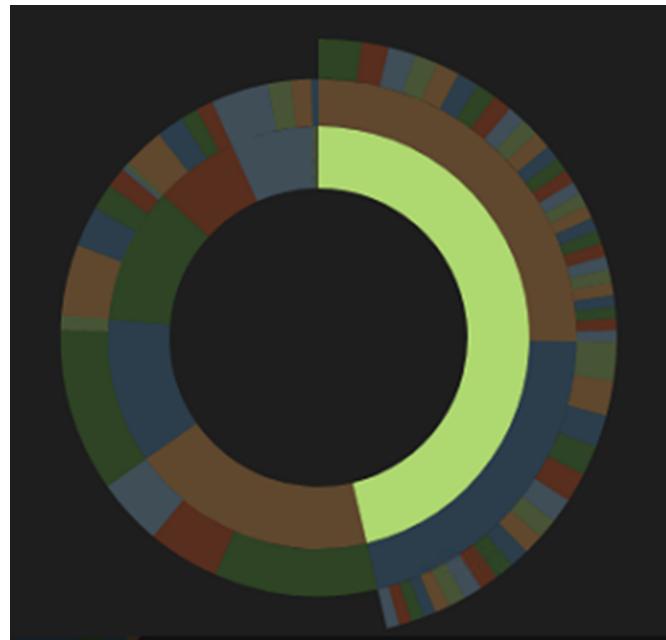
- Code มีจำนวนทั้งหมด 3,344 บรรทัด คิดเป็น 65%
- Comment มีจำนวนทั้งหมด 679 บรรทัด คิดเป็น 13%
- Blank มีจำนวนทั้งหมด 1,157 บรรทัด คิดเป็น 22%



ภาพ Line Breakdown

1.2 Directory Structure

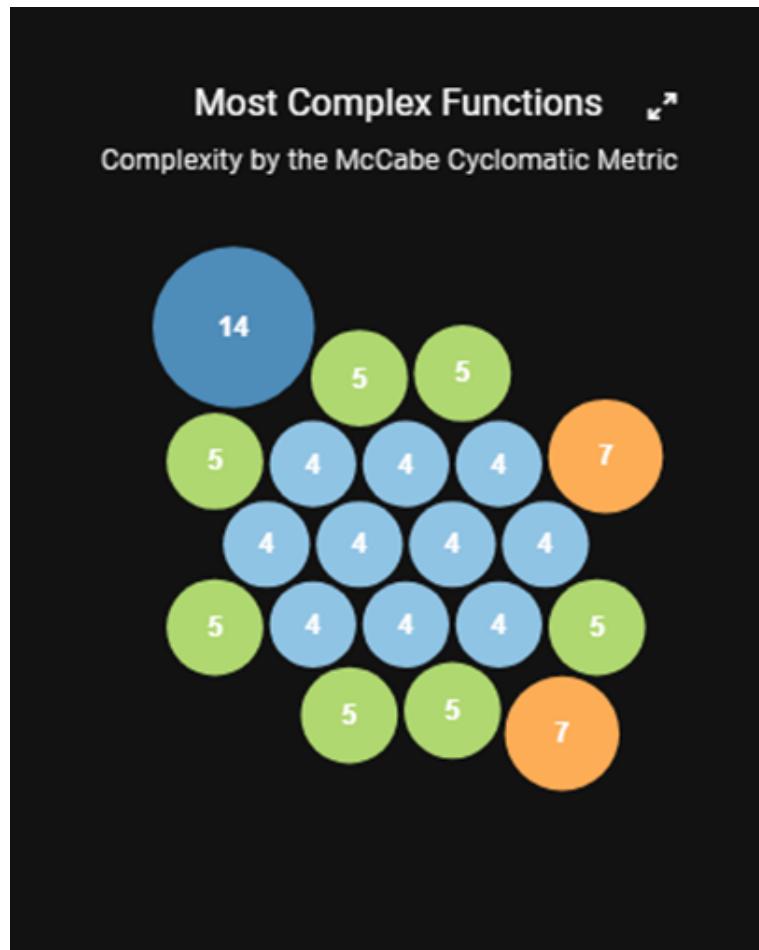
โครงสร้างของ Contested Multiplayer Online Game จะมีโครงสร้างหลักๆ อยู่ 5 ส่วนก็คือ networking(เขียวอ่อน), core(เหลือง), db(น้ำเงิน), model(เขียวเข้ม), utility(สีแดง), metadata(สีฟ้า) ซึ่งสัดส่วนของ Networking จะมีทั้งหมด 46.3% ของโปรเจคซึ่งจะมี Request และ Response เป็นหลักๆ



ภาพ Directory Structure

1.3 Most Complex Functions

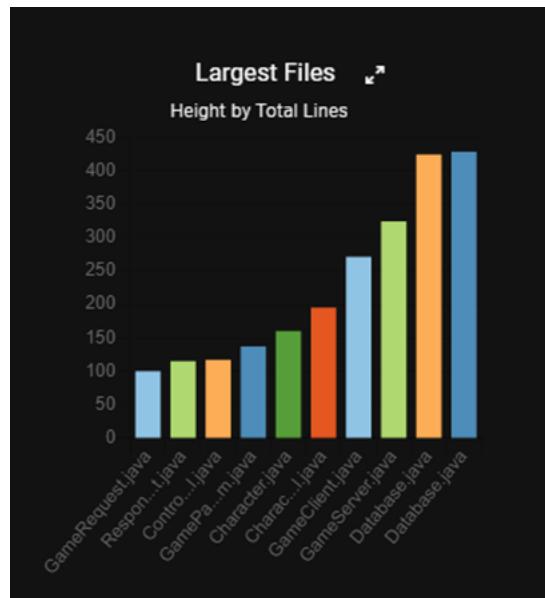
วิเคราะห์โดยใช้ Cyclomatic metric ซึ่งจะนับจากการใช้จุดตัดสินใจ เช่น คำสั่ง if else while for โดย demo_data.main จะมีความซับซ้อนมากที่สุดคือ 14



ภาพ Most Complex Functions

1.4 Largest Files

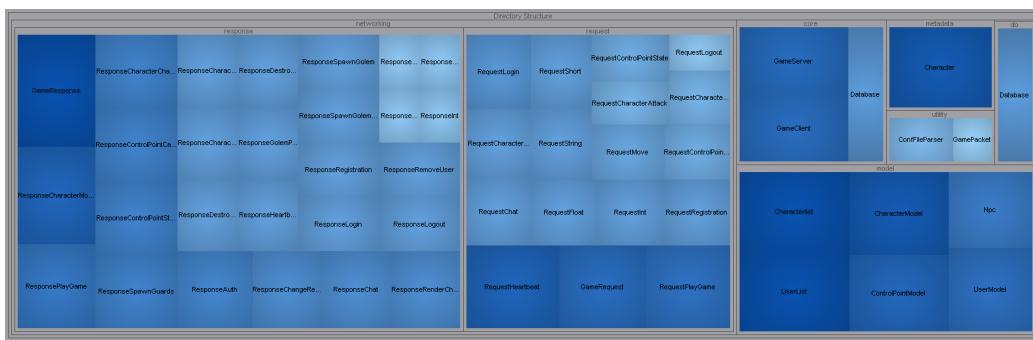
แสดงจำนวนบรรทัดที่รวมกันทั้งหมดในแต่ละไฟล์ซึ่งไฟล์Databaseจะมีจำนวนมากที่สุด



ການ Largest Files

1.5 Percent Lack Of Cohesion

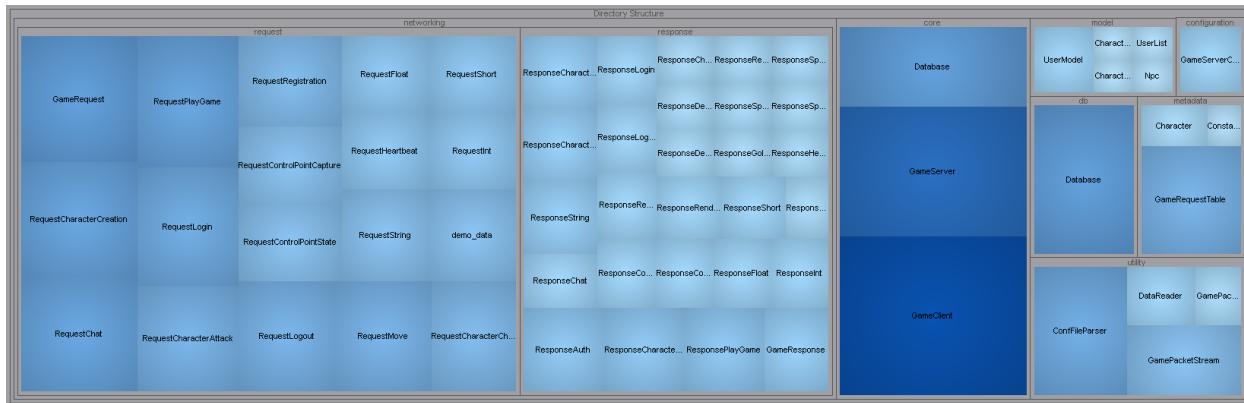
Percent Lack Of Cohesion คือการบอกถึงความสอดคล้องกันของการทำงานโดยเปอร์เซ็นต์ที่ต่ำกว่าหมายถึงการทำงานร่วมกันระหว่างข้อมูลและวิธีการในชั้นเรียนที่สูงขึ้นจากการเพราะนั้น Class ควรจะมีเปอร์เซ็นต์ที่น้อย การทดสอบในระดับ Class จะแสดงให้เห็นว่า Percent Lack of Cohesion ที่น้อยที่สุดเกือบ 20%



gap การวิเคราะห์ Percent Lack Of Cohesionโดยแผนภูมิทรีแมป

1.6 Count Class Coupled

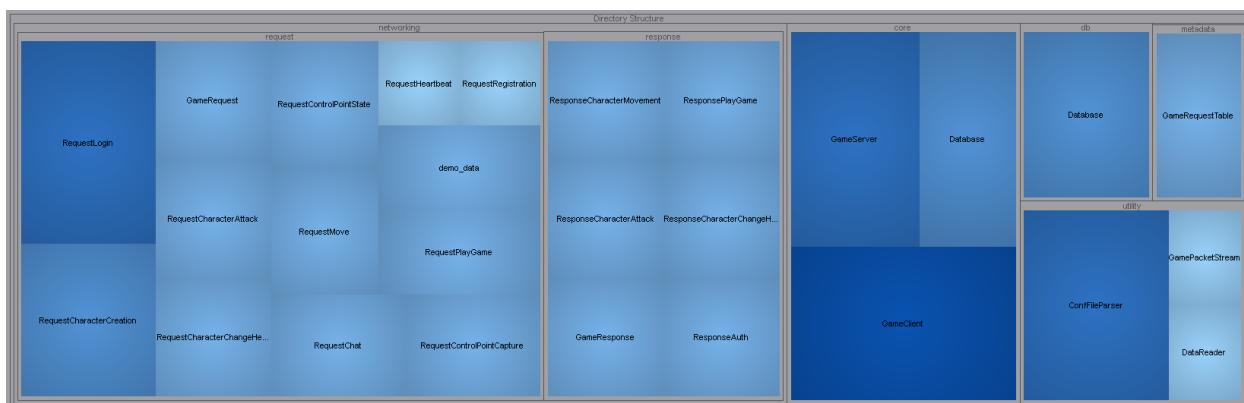
Count Class Coupled คือ ระดับความเกี่ยวข้อง หรือ พึ่งพา กันของ Module ต่างๆ ในระบบ
จากการทดสอบในระดับ Class จะแสดงให้เห็นว่า GameClient จะมีจำนวนมากที่สุดโดยจะซึ่งมีจำนวนมาสูง 22



ภาพ การวิเคราะห์ Count Class Coupled โดยแผนภูมิทรีแมป

1.7 MaxNesting

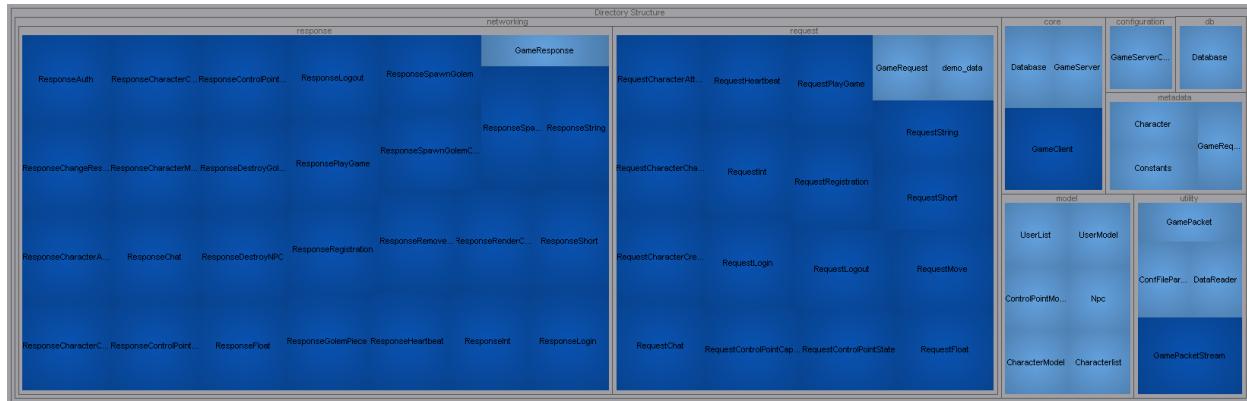
MaxNesting คือ ระดับการซ่อนสูงสุดของโครงสร้าง เช่น if, while, for ,switch ในฟังก์ชันการวิเคราะห์ MaxNesting ระดับ Class จะแสดงให้เห็นว่า GameClient จะเสียที่ใช้มากที่สุดที่จำนวน 5



ภาพ การทดสอบ Max Nesting โดยแผนภูมิทรีเมป

1.8 MaxInheritanceTree

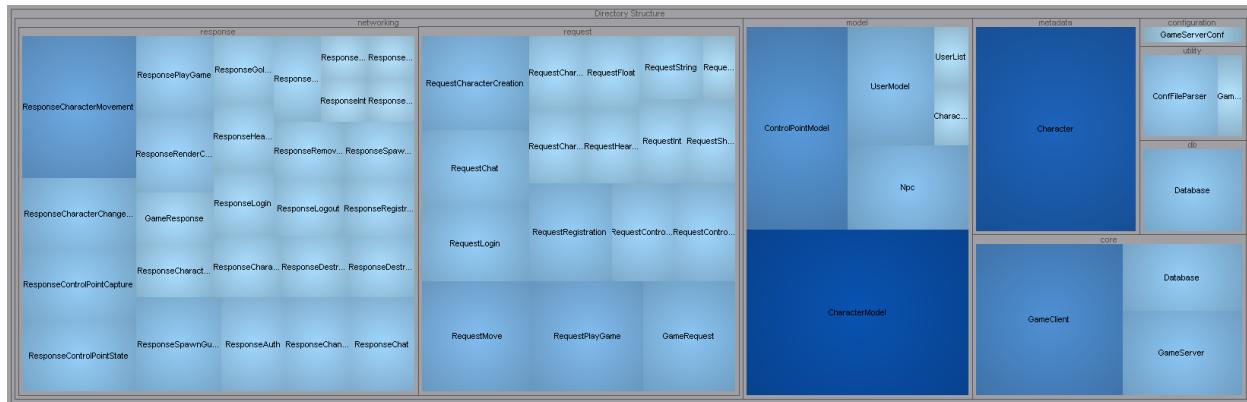
MaxInheritanceTree คือ ความลึกของคลาสภายในลำดับชั้นการสืบทอดคือจำนวนสูงสุดของโหนดคลาสไปยังรากของแผนผังการสืบทอด โหนดรูทมี Depth of Inheritance Tree เป็น 0 ยิ่งอยู่ในลำดับชั้นลึกเท่าไหร่ ยิ่งความสามารถสืบทอดเมื่อต้องได้มากเท่านั้น ซึ่งเพิ่มความซับซ้อนขึ้น โดยจากภาพจะเป็นการทดสอบในระดับ Class ซึ่งส่วนมาก Max Inheritance Tree จะมีค่าเท่ากับ 2



ภาพ Inheritance Tree

1.9 CountDeclInstanceVariable

จำนวนตัวแปรอินสแตนซ์ตัวแปรที่กำหนดในคลาสที่เท้าถึงได้ผ่านวัตถุของคลาสนั้นเท่านั้น โดยจากการทดสอบในระดับ Class ซึ่ง CharacterModel ที่อยู่ในส่วนของ model จะมีตัวแปรทั้งหมด 18 ตัวและ Character ที่อยู่ในส่วนของ metadata จะมีตัวแปรทั้งหมด 16 ตัว



ภาพ CountDeclInstanceVariable

2. JArchitect

JArchitect คือ เครื่องมือสำหรับการทดสอบแบบ static analysis ของภาษา Java โดยเครื่องมือนี้รองรับการใช้งานเพื่อตัด code matrix ออกทั้งปัจจัยสามารถสร้างภาพของ dependencies ได้โดยใช้ directed graphs และ dependency matrix

ผลการวิเคราะห์ด้วย JArchitect

Project File Analysis

The screenshot shows the 'Project Rules' section of the JArchitect Project File Analysis. It lists 401 queries across several categories:

- Project Rules (401 queries)**
 - Quality Gates (14 queries)**
 - Code Smell (8 queries)**
 - Code Smells Regression (9 queries)**
 - Object Oriented Design (4 queries)**
 - Architecture (6 queries)**
 - Design (59 queries)**
 - UML Design Validation (11 queries)**
 - Import Statements (6 queries)**
 - Optimization (11 queries)**
 - Strict Exceptions (12 queries)**
 - String and StringBuffer (15 queries)**
 - API Breaking Changes (9 queries)**
 - Code Coverage (13 queries)**
 - Dead Code (5 queries)**
 - Controversial (21 queries)**
 - Empty Code (11 queries)**
 - Unnecessary Code (8 queries)**
 - Logging (9 queries)**
 - Basic (24 queries)**
 - Naming Conventions (11 queries)**

> **Trend Metrics (75 queries)**

 - HotSpots (7 queries)**
 - Code Diff Summary (25 queries)**
 - Defining JustMyCode (4 queries)**
 - Statistics (12 queries)**
 - Samples of Custom rules (12 queries)**

Rules extracted from Source Code (0 query)

ภาพจากการวิเคราะห์ไฟล์ Contested Multiplayer Online Game ด้วย JArchitect

2.1 Quality Gates

Quality Gates

	Fail	2
	Warn	0
	Pass	3

ภาพ Quality Gates จากการวิเคราะห์

Quality Gates คือ ชุดของ metric ที่ต้องทำให้สำเร็จถึงจะทำให้ Quality Gates ผ่านเกณฑ์ โดยแบ่งเกณฑ์ออกเป็น 3 แบบคือ ผ่าน (Pass), เตือน (Warn), ไม่ผ่าน(Fail)

Passed conditions							
Quality Gates	Blocker issues	Duplicated lines	New critical issues	Public documented API	Technical debt	Technical debt ratio	Test coverage
Low	= 0	<50%	<20	>25%	<60d	<40%	>10%
Standard	= 0	<25%	<10	>50%	<30d	<20%	>25%
Hard	= 0	<15%	<0	>60%	<10d	<20%	>70%

ภาพเกณฑ์ในการวิเคราะห์ Quality Gates

Active	#Items	Code Queries and Rules
<input checked="" type="checkbox"/>	0	Quality Gates Evolution
<input checked="" type="checkbox"/>	N/A %	Percentage Code Coverage
<input checked="" type="checkbox"/>	N/A %	Percentage Coverage on New Code
<input checked="" type="checkbox"/>	N/A %	Percentage Coverage on Refactored Code
<input checked="" type="checkbox"/>	0 issues	Blocker Issues
<input checked="" type="checkbox"/>	0 issues	Critical Issues
<input checked="" type="checkbox"/>	0 issues	New Blocker / Critical / Major Issues
<input checked="" type="checkbox"/>	2 rules	Critical Rules Violated
<input checked="" type="checkbox"/>	19.65 %	Percentage Debt
<input type="checkbox"/>	6.5 man-days	Debt
<input checked="" type="checkbox"/>	N/A man-days	New Debt since Baseline
<input checked="" type="checkbox"/>	3 packages	Debt Rating per Package
<input type="checkbox"/>	12.83 man-days	Annual Interest
<input checked="" type="checkbox"/>	N/A man-days	New Annual Interest since Baseline

ภาพรายละเอียดภายใน Quality Gates

2.1.1 Critical Rules Violated

Avoid types too big	1 issue
Avoid methods with too many parameters	1 issue

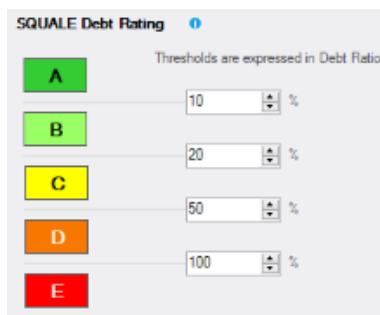
Avoid types too big คือ การที่ class ได class หนึ่งมีจำนวน NOC มากกว่า 200 บรรทัดขึ้นไปซึ่งนั้นทำให้เกิดความซับซ้อนต่อการพัฒนาและบำรุงรักษา

Avoid methods with too many parameters คือ ในหนึ่ง method มีมากกว่า 8 parameters ซึ่งทำให้ยากต่อการเรียกใช้และอาจทำให้ประสิทธิภาพต่ำลงด้วย

2.1.2 Debt Rating per Package

3 packages	debtRating	debtRatio	devTimeIn	debtInMa	issues
3 packages matched					
Contested_Massive_Multiplayer_0	D	43.38	7d 4h	3d 2h	214 issues
core	D	22.06	2d 6h	5h 3min	26 issues
metadata	D	20.01	4d 2h	6h 53min	53 issues
model	D				

Debt Rating per Package คือการประมาณค่าของ Debt Ratio โดยแบ่งจาก E ถึง A โดยจะแบ่งตามการตั้งค่าของ JArchitect



ภาพการแบ่งเกณฑ์ของ Debt Rating

Debt Ratio < 10% = A

Debt Ratio < 20% = B

Debt Ratio < 50% = C

Debt Ratio < 100% = D

Debt Ratio > 100% = E

Debt Ratio คือ อัตราส่วนของ Debt Amount เปรียบเทียบกับ estimated effort to develop the code element

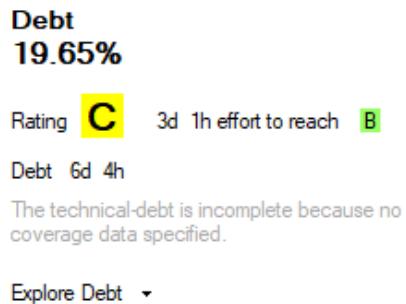
Debt amount คือ จำนวนปัญหา (issues) ทั้งหมด Estimated effort to develop the code element คือ การอนุมานค่าประมาณการของกำลังคนในการเขียนโค้ด 1000 บรรทัด

2.2 Code Smell

Code Smell หมายถึง Source code ของโปรแกรมนั้นสามารถบ่งชี้ได้ว่าเป็นปัญหาได้ โดย code smell นั้นไม่ใช่บัค หรือ การทำงานผิดพลาดของโปรแกรมแต่มันคือ จุดอ่อนในการออกแบบที่อาจจะส่งผลให้การพัฒนาของโปรแกรมช้าลง หรือ สามารถเพิ่มบัคของโปรแกรมได้ในอนาคตโดยการจะดูว่า code มี smell สามารถบ่งบอกได้จาก technical debt หรือเรียกอีกชื่อว่า “หนี้ทางเทคนิค”

#Items	Code Queries and Rules
1	Avoid types too big
0	Avoid types with too many methods
0	Avoid types with too many fields
0	Avoid methods too big, too complex
1	Avoid methods with too many parameters
0	Avoid methods with too many local variables
5	Avoid methods potentially poorly commented
3	Avoid types with poor cohesion

2.2.1 Technical debt



ภาพจาก การวิเคราะห์ผลจากโปรแกรม JArchitect

Technical debt คือ ค่าการทำงานในการแก้ไขปัญหา โดยเกิดจากการลาก่อน หรือ การออกแบบที่ไม่เหมาะสมกับงานทำให้ต้องกลับมาแก้ไขปัญหาซ้ำๆ ซึ่งหากไม่แก้ไขปัญหาอาจจะทำให้เกิดปัญหาอื่นๆ ได้

Avoid methods potentially poorly commented คือ กฎเกณฑ์ของโปรแกรม JArchitect ว่าในหนึ่ง method มี comment น้อยกว่า 20 บรรทัดควรที่จะ comment

Avoid types with poor cohesion นั้นมีพื้นฐานมาจาก LCOM code metric โดยขึ้นอยู่กับจำนวน method ที่มีการเรียกใช้ ตัวแปร ที่มีการประกาศไว้ใน class ซึ่งหาก method เรียกใช้ตัวแปรที่ประกาศไว้ในคลาสมากก็จะทำให้เกิด Low หรือ poor cohesion ซึ่งหากใช้น้อยก็จะเกิด high cohesion หรือความชับช้องน้อยลงตามหลัก High Cohesion & Low Coupling

2.3 Architecture

Active	#Items	Code Queries and Rules
<input checked="" type="checkbox"/>	0	Avoid packages dependency cycles
<input checked="" type="checkbox"/>	1	Projects with poor cohesion (RelationalCohesion)
<input checked="" type="checkbox"/>	1	Packages with poor cohesion (RelationalCohesion)
<input checked="" type="checkbox"/>	0	Projects that don't satisfy the Abstractness/Instability principle
<input type="checkbox"/>	7	Higher cohesion - lower coupling
<input type="checkbox"/>	0	Example of custom rule to check for dependency

Projects with poor cohesion คือ กฎของ Relational Cohesion metric ว่าด้วย Relation ใน Project ที่ควรเรียกใช้ class ให้น้อยเพื่อเพิ่ม High Cohesion

Package with poor cohesion คือ กฎของ Relational Cohesion metric ว่าด้วย Relation ใน Package ที่ควรเรียกใช้ class ให้น้อย เพื่อเพิ่ม High Cohesion

เปรียบเทียบผลลัพธ์ระหว่าง Understand และ JArchitect

1. Reusability การวัดการนำกลับมาใช้ใหม่ของโค้ด

1.1 Inheritance Tree

2. Complexity ความซับซ้อนของโค้ด

2.1 Cyclomatic Complexity

2.2 Nesting Depth

3. Understandability การรับความเข้าใจของโค้ด

3.1 Percentage of Comment

3.2 Count Instant variable

4. Maintainability การบำรุงรักษา

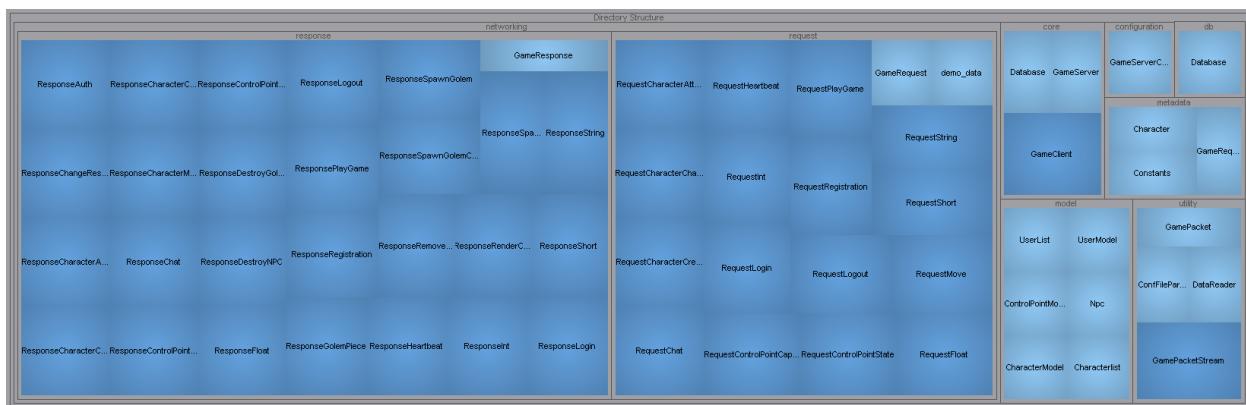
4.1 Line of code

4.2 Lack of Cohesion

1. Reusability

1.1 Inheritance Tree

1.1.1 Understand ทดสอบในระดับ Class ที่ส่วนมาก Max Inheritance จะมีค่าเท่ากับ 2 unit

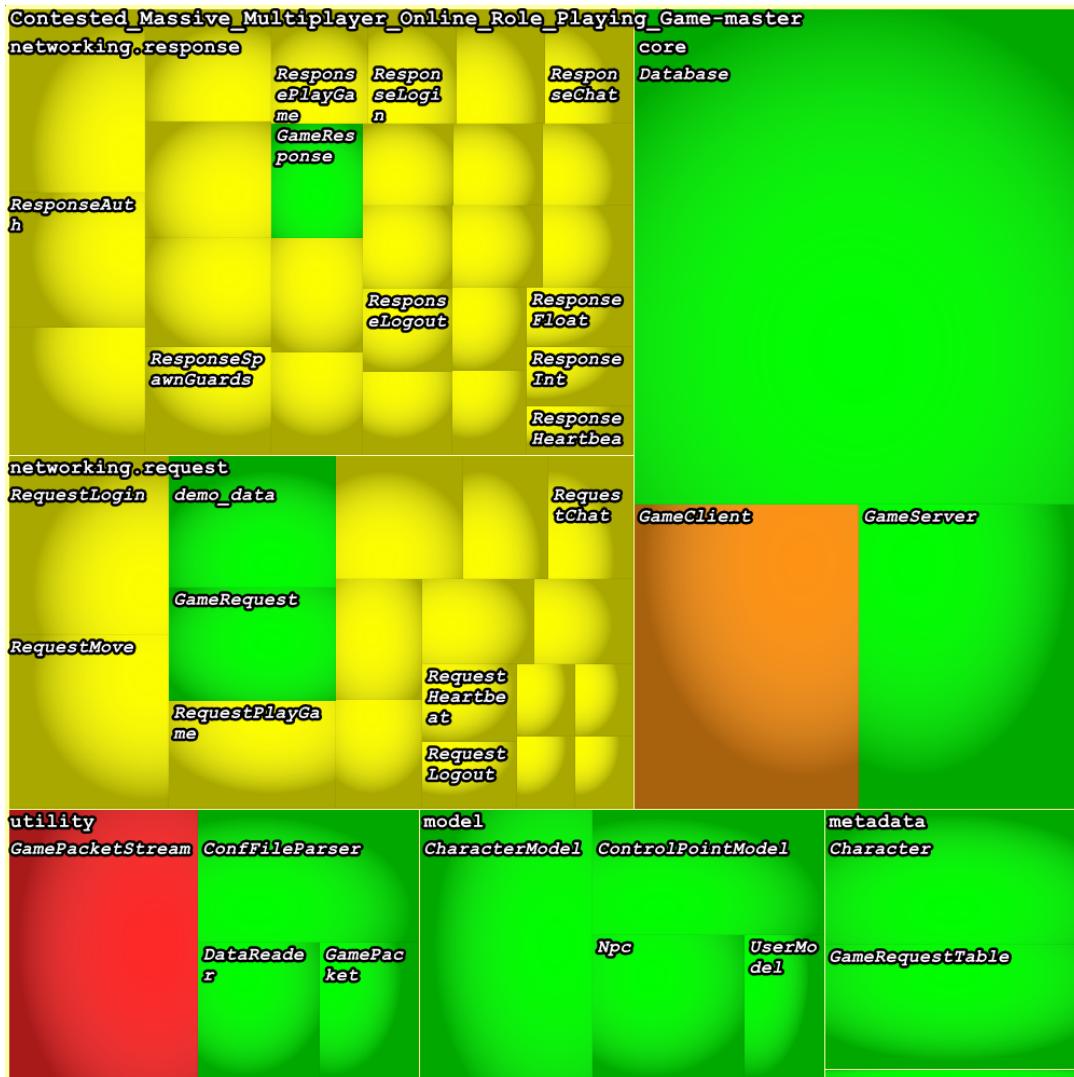


ภาพ Max Inheritance

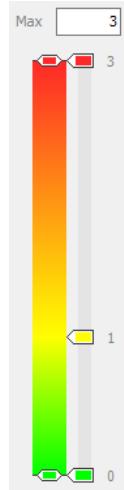


ภาพ Max ของ Max Inheritance

1.1.2 JArchitect ทำการทดสอบด้วยโปรแกรม JArchitect ได้ผลลัพธ์ดังนี้



จากการพบร่างส่วนของ GamePacketStream นั้นมี 3 unit ซึ่งเป็นตัวสูงสุดของผลลัพธ์โดยเกณฑ์ทั้งต่อไปนี้



ภาพเกณฑ์การวัด Depth of inheritance โดยใช้โปรแกรม JArchitect

สรุป Reusability

จากการทดสอบ Understand สามารถตรวจสอบ Inheritance ได้สูงสุดเท่ากับ 2 โดยที่ส่วนใหญ่จะมีค่าเท่ากับ 2 ผิดกับ Jarchitect ที่สามารถตรวจสอบได้ค่าสูงสุดเท่ากับ 3 โดยที่ Class ส่วนอื่น ๆ มีค่าเท่ากับ 1 ไม่กี่ 0 จึงสรุปได้ว่าプロジェクトที่นำมาตรฐานสามารถ Reusability หรือนำมากตั้งมาใช้ใหม่ได้ดี

2.Complexity

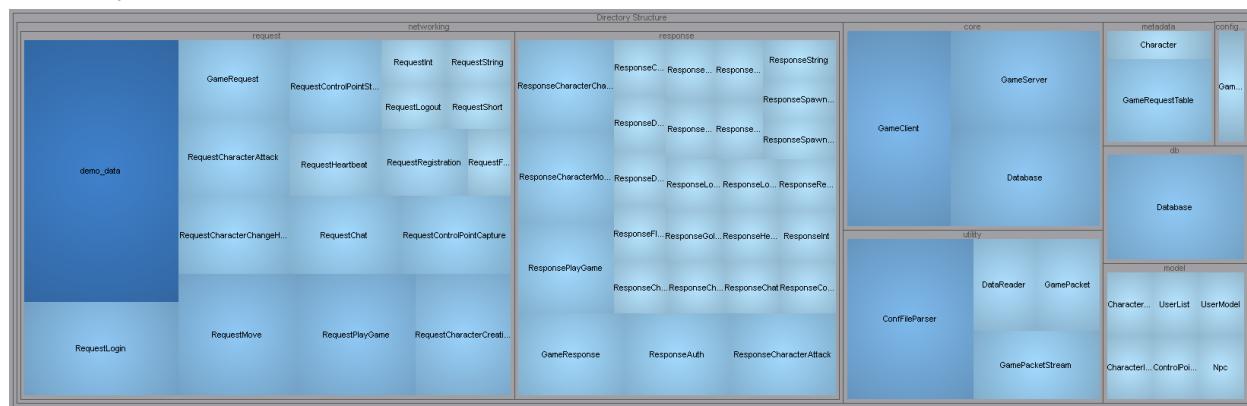
2.1 Cyclomatic

2.1.1 Understand

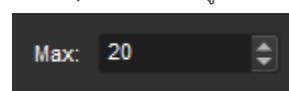
ทดสอบในระดับ Class โดยมี demo_data ที่อยู่ในส่วนของ request ได้ 14 unit

GameClient ที่อยู่ในส่วนของ core ได้ 7 คะแนน ConfFileParser ที่อยู่ในส่วนของ utility ได้ 7 unit

และส่วนอื่นอยู่ในช่วง 1-5 unit

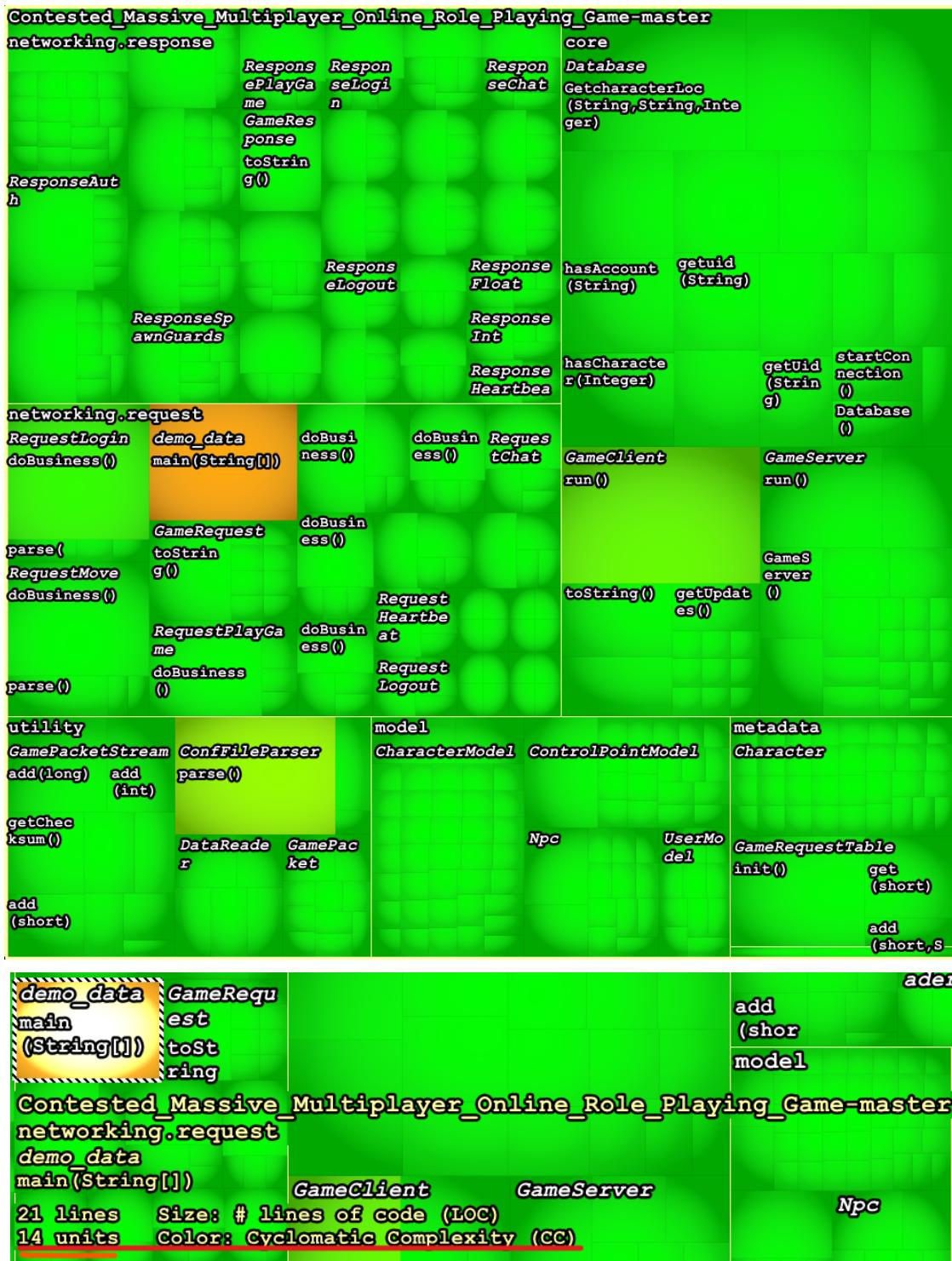


ภาพ Cyclomatic แผนภูมิที่มีสี

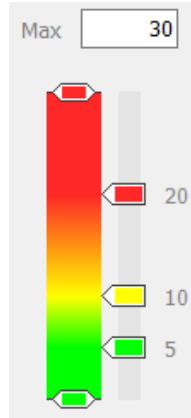


ภาพ กำหนด Max ของ Cyclomatic

2.1.2 JArchitect นั้นมีบาง metrics ที่สามารถแสดงเป็น code metrics view ได้และ แสดงผลลอก
เป็นตัวจงภาพนี้



จากภาษาจะเห็นได้ว่าในส่วนของ demo_data main(String[]) นั้นจะขึ้นเป็นสิ่งเรียกใช้โดยทันทีที่มีอยู่ทุกชื่อใน class นี้สูง โดยการรับตัวตามเกณฑ์ที่ตามมาเพื่อไปใช้

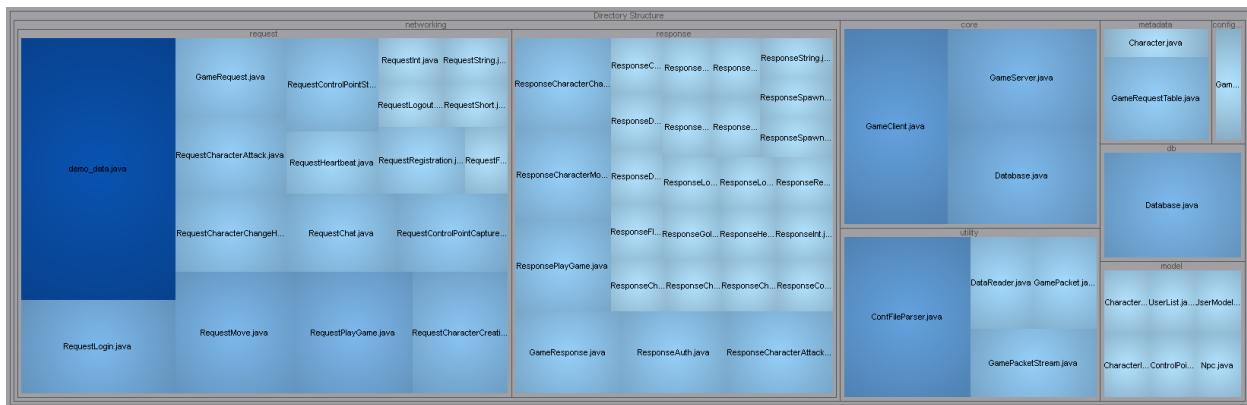


ภาพการวัดเกณฑ์ของ Cyclomatic Complexity (CC) ของ JArchitect

2.2 Nesting Depth

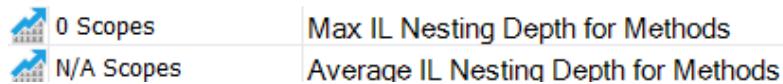
2.2.1 Understand สามารถทดสอบในระดับ Class

1. GameClient ที่อยู่ในส่วน core มีความทับซ้อนกันที่ 5
 2. GameServer ที่อยู่ในส่วนของ core จะมีความทับซ้อนกันที่ 4
 3. ConFileParser ที่อยู่ในส่วนของ utility จะมีความทับซ้อนกันที่ 4
 4. RequestLogin ที่อยู่ในส่วนของ request จะมีความทับซ้อนกันที่ 4



ภาพ การวิเคราะห์ Cyclomatic Matric ในระดับ Class โดยแผนภูมิทรีเมป์

2.2.2 JArchitect ไม่สามารถวัด nesting depth ได้เนื่องจากจำกัดของโปรแกรมหรือ เกิดขึ้นจากตัวไฟล์โปรแกรมที่ทดสอบไม่สามารถวัดได้ จึงแสดงภาพดังต่อไปนี้



Max IL Nesting Depth for Methods 0 Scopes

Average IL Nesting Depth for Methods N/A Scopes

สรุป Complexity

1. Cyclomatic Complexity

Understand และ Jarchitect ให้ผลลัพธ์ว่า Class Demo_main มีการใช้ Decision point ที่ค่อนข้างเยอะ

2. Nesting Depth

Understand ให้ผลลัพธ์ว่า GameClient มีความทับซ้อนกันมากที่สุดเท่ากับ 5 ชั้น เกินกว่าค่ากำหนดที่ได้ตั้งไว้และโดยภาพรวมของโปรแกรมมี Nesting Depth ที่ค่อนข้างจะสูง

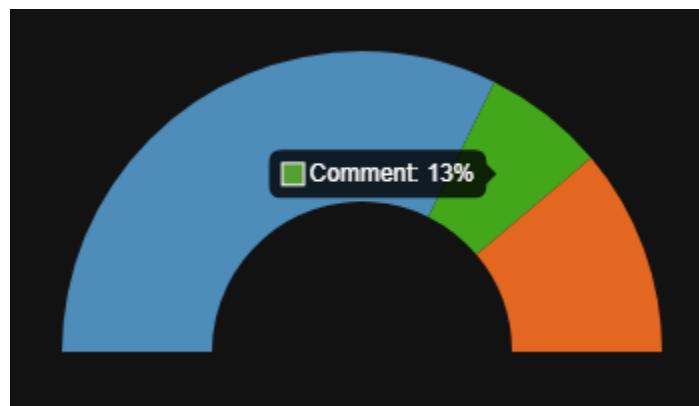
Jarchitect ไม่สามารถที่จะตรวจสอบ Nesting Depth ได้
ดังนั้น Complexity จึงค่อนข้างที่จะซับซ้อน

3. Understandability

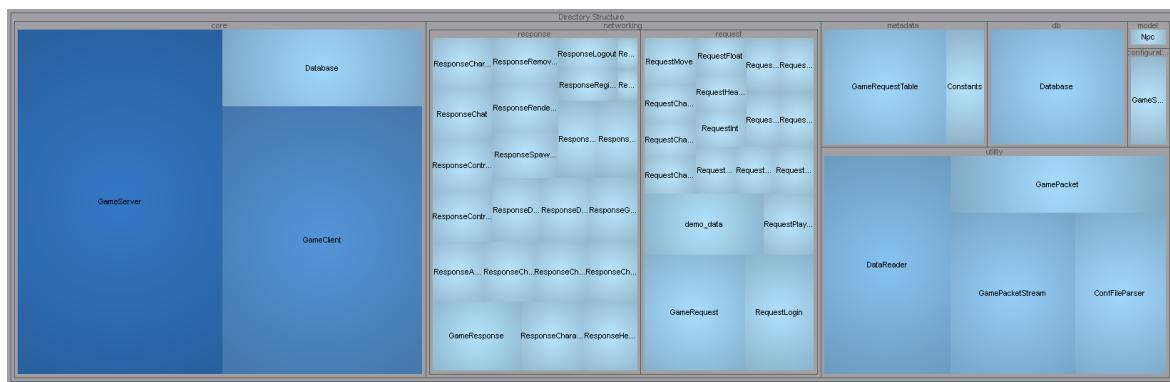
3.1 Count Line Comment

3.1.1 Understand ทดสอบจำนวน Comment ในแต่ละ Class โดยใช้ Count Line Comment ซึ่งได้ค่ามากที่สุดโดยเลือกมา 3 อันดับแรก

1. GameServer จำนวน Comment 115 บรรทัด
2. GameClient จำนวน Comment 87 บรรทัด
3. DateReader จำนวน Comment 45 บรรทัด



ภาพ แสดงเปอร์เซ็นของคอมเม้น

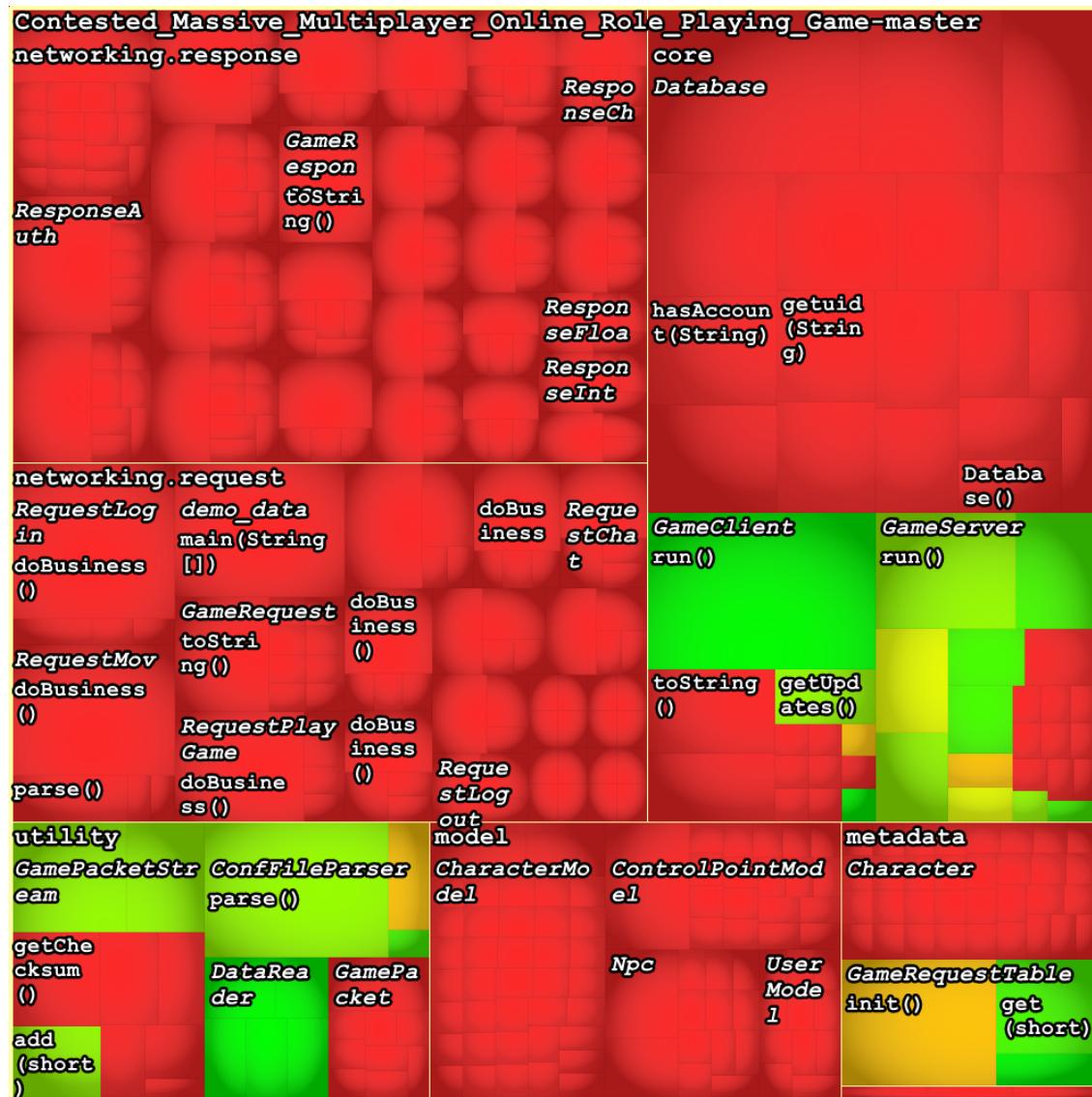


ภาพ การวิเคราะห์ Count Line Comment

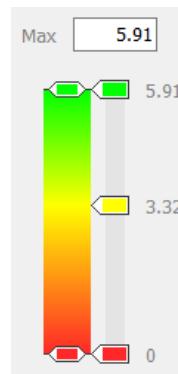
3.1.2 JArchitect วัดค่าของ comment ได้ดังภาพต่อไปนี้

Comment
4.68%

ภาพสถิติ comment



ภาพจุดที่แสดงถึงการ comment ภายในโปรแกรมที่ทดสอบ

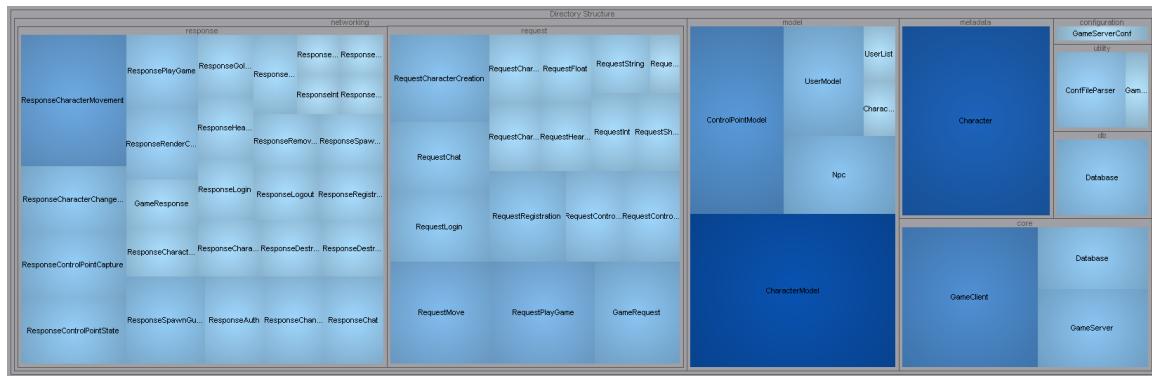


ภาพการวัดเกณฑ์ของ comment ของ JArchitect

ชี้เงินที่นี่ก็จะจากการปรับค่าของผู้จัดทำเงื่อนไขจาก default ของโปรแกรมนั้นยกต่อการดูแลสังเกต ทำให้จำเป็นต้องปรับเปลี่ยน

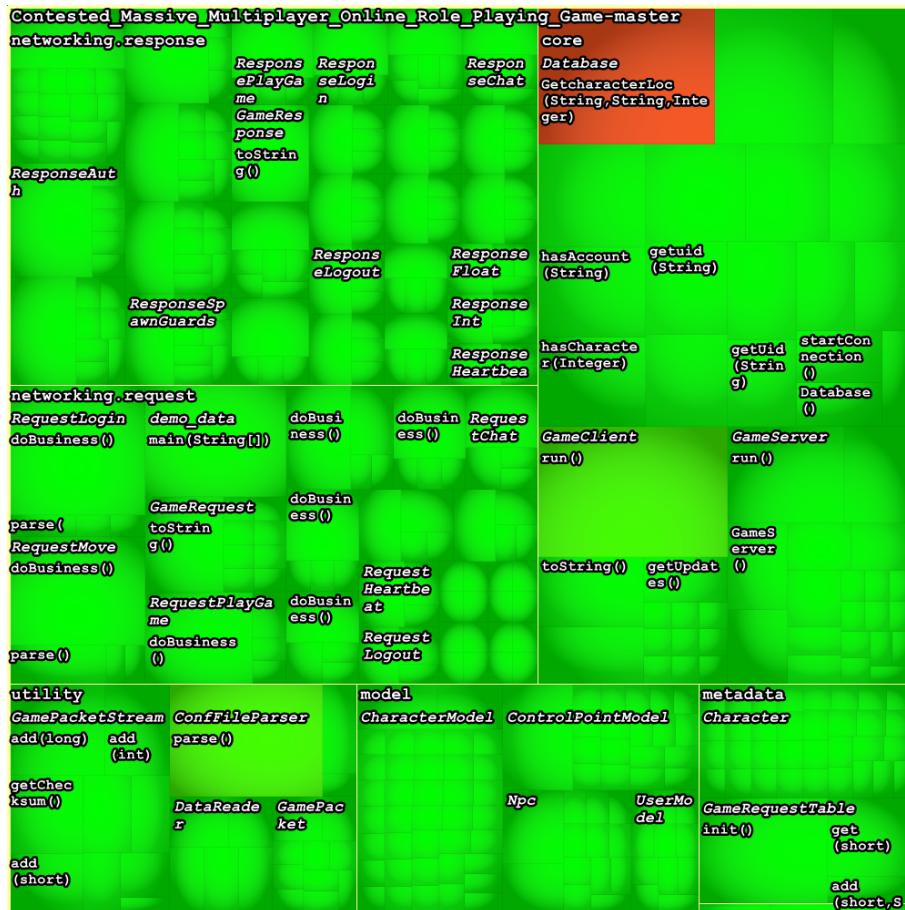
3.2 Count Instance variable

3.2.1 Understand จำนวนตัวแปรอินสแตนซ์ - ตัวแปรที่กำหนดในคลาสที่เข้าถึงได้ผ่านวัตถุของคลาสนั้นเท่านั้น โดยจากการเป็นการทดสอบในระดับ Class ซึ่ง CharacterModel ที่อยู่ในส่วนของ model จะมีตัวแปรทั้งหมด 18 ตัวและ Character ที่อยู่ในส่วนของ metadata จะมีตัวแปรทั้งหมด 16 ตัว



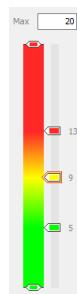
ກາພ ກາຣົວເຄຣະໜີ Count Instance Variable

3.2.2 JArchitect Count Instance variable คือ จำนวนตัวแปรภายในคลาสๆ หนึ่ง โดยหากมีค่าสูงจะส่งผลต่อ object ภายในคลาส ทำให้ยากต่อการรับการ



ภาพผลลัพธ์ของการทดสอบ Count Instance variable

จากการจะเห็นว่าผลลัพธ์ที่ได้มานั้นภายใน Database มีค่า 12 variable ซึ่งเป็นค่าสูงสุดของผลลัพธ์โดยเกณฑ์การวัดใช้ความภาพดังต่อไปนี้



ภาพเกณฑ์การวัด Count Instant variable โดยใช้โปรแกรม JArchitect

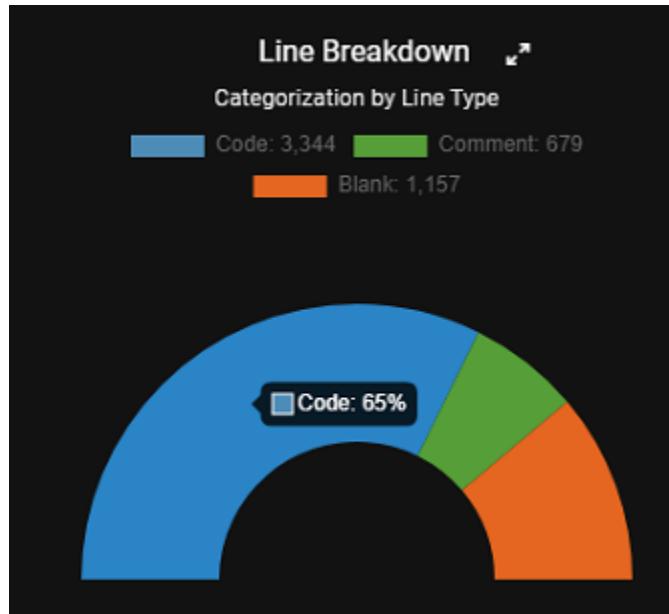
สรุป Understandability

จากการทดสอบพบว่า Percentage of Comment ทั้งสองโปรแกรมทดสอบออกมาแล้วว่ามี POC น้อยมากในภาพรวมทำให้การพัฒนาต่อได้ยาก แต่ในส่วนของ Count Instant variable นั้นมีเพียงบางจุดที่มีค่าสูงแต่ในภาพรวมถือว่าจัดการ variable ได้ในระดับกลาง

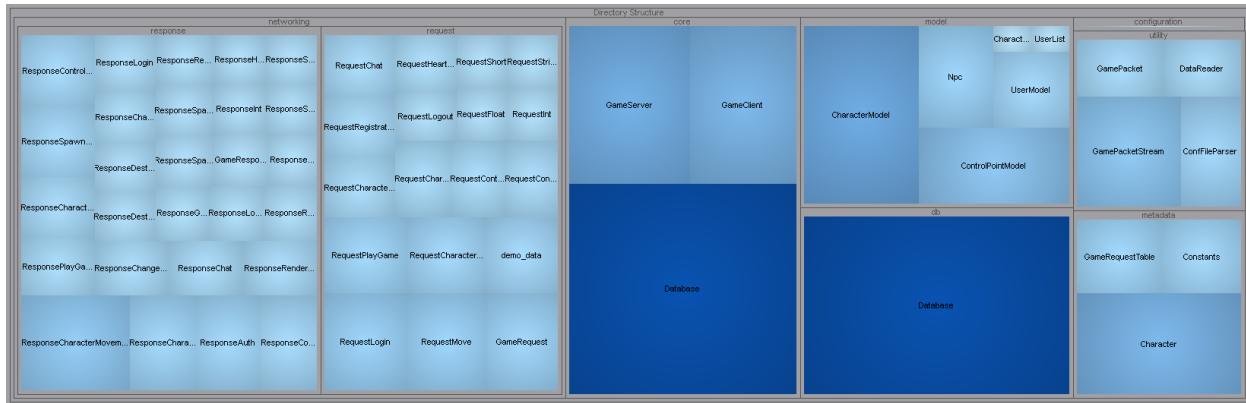
4. Maintainability

4.1 Line of code

4.1.1 Understand จะแสดงให้เห็นเป็นสัดส่วนของโค้ดโดยจะแสดงเป็นเบอร์เข็นและจำนวนตัวเลขของมาและแสดง Line of Code ในระดับ Class ที่มีจำนวนมากที่สุด ซึ่ง database ในส่วนของ core มีทั้งหมด 337 และในส่วนของ db มีจำนวน 336



ภาพ Line Breakdown โดยแสดงเบอร์เข็น Code



ภาพ แสดงLine of Code ในระดับ Class โดยใช้แผนภูมิที่แม่ปี

4.1.2 JArchitect นับ LOC จาก concrete ทำให้จำนวนบรรทัดนั้นอยู่กว่าโปรแกรมที่ใช้ static test ตัวอื่นๆ

Interfaces, abstract methods and enumerations have a LOC equals to 0. Only concrete code that is effectively executed is considered when computing LOC.

ภาพอธิบายการคำนวณ LOC ของ JArchitect

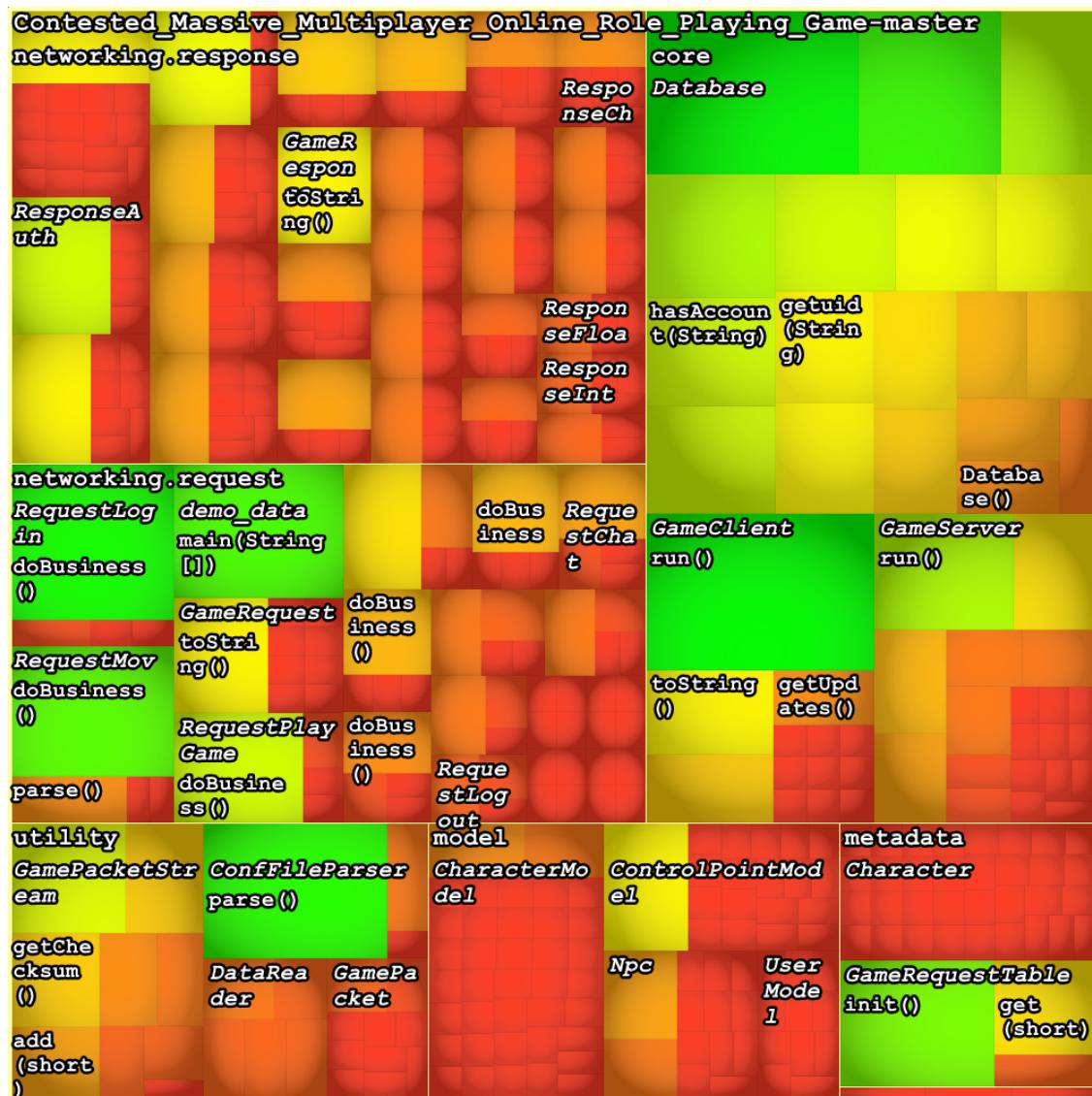
Logical lines of Code

1 099

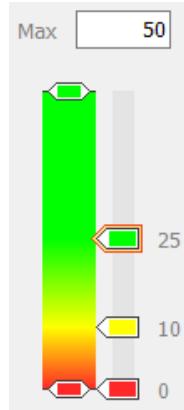
0 (NotMyCode)

Estimated Dev Effort 33d

ภาพสถิติของ LOC



ภาพแสดงจำนวน LOC ของแต่ละ method



ภาพการวัดเกณฑ์ของ Line of Code (LOC) ของ JArchitect

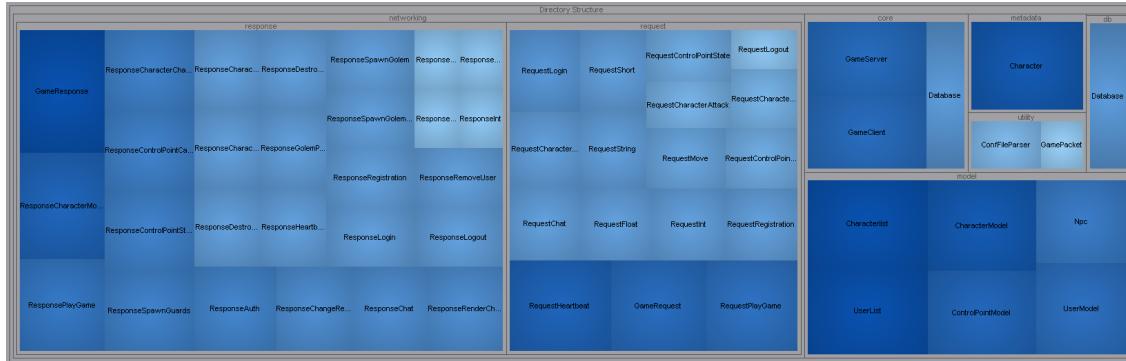
4.2 Lack of Cohesion

4.2.1 Understand

การทดสอบในระดับ Class จะแสดงให้เห็นว่า Percent Lack of Cohesion ที่น้อยที่สุด 6 อันดับคือ

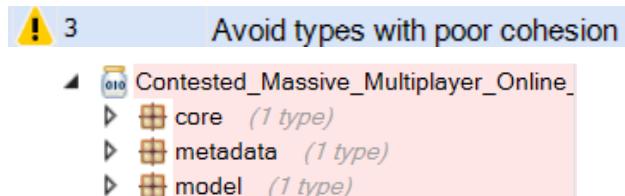
1. GamePacket มีเปอร์เซ็นเท่ากับ 20 %
2. ResponseString มีเปอร์เซ็นเท่ากับ 25 %
3. ResponseShort มีเปอร์เซ็นเท่ากับ 25 %
4. ResponseInt มีเปอร์เซ็นเท่ากับ 25 %
5. ResponseFloat มีเปอร์เซ็นเท่ากับ 25 %
6. RequestLogout มีเปอร์เซ็นเท่ากับ 25 %
7. ส่วนอื่นๆ มีภาพรวม Percent Lack Of Cohesion ที่ค่อนข้างสูงซึ่งแสดงว่าการทำงานของ Class ในแต่ละ Class มีการทำงานที่สอดคล้อง

กันค่อนข้างน้อย

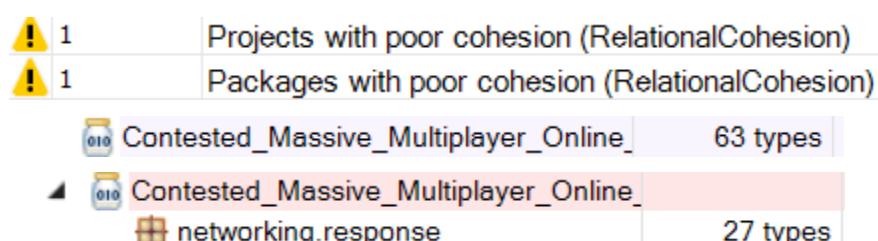


ภาพ การวิเคราะห์ Percent Lack of Cohesion

4.2.2 JAchitect นั้นไม่ได้แสดงในรูปแบบสรุปผลให้แต่จะบ่งชี้ว่าโค้ดส่วนไหนที่ควรปรับแก้



ภาพ type poor cohesion ที่เกิดขึ้นใน code smell



ภาพ project, package poor cohesion ที่เกิดขึ้นใน architecture

สรุป Maintainability

จากการทดสอบสรุปได้ว่า LOC จะมีค่าสูงเมื่อออยู่ใน class ที่มีความสำคัญมากๆ เช่น Database, GameClient ทำให้การบำรุงรักษาจะทำได้ยากในบางชุดส่วน LCOM metric นั้นใน JArchitect ผลลัพธ์ไม่ได้ออกมาในรูปของตัวเลขแต่เป็นบวกเพียงจุดที่ควรนำไปแก้ไข ส่วนใน Understand จะแสดงในรูปของทรีแมพโดยผลกระทบที่ได้พนับว่าค่าของ LCOM จะสูงเมื่อออยู่ใน class ที่มีการทำงานร่วมกันสูง

ข้อแตกต่างของ Understand และ JArchitect

Understand

ข้อดี

- ในแต่ละการวัด matrix จะมองได้เจ้ายempre มีการอธิบายในโปรแกรม
- UX/UI ใช้งานได้เจ้าย
- มีภาพรวมของโปรแกรมที่อ่านเข้าใจได้เจ้ายบอกเป็นสัดส่วนชัดเจนว่าโปรแกรมมีส่วนอะไรบ้าง

ข้อเสีย

- บอกจุดบกพร่องได้ไม่ชัดเจนเท่า JArchitect
- หากใช้งานเกิน 14 วันจำเป็นต้องเสียค่าใช้จ่าย

JArchitect

ข้อดี

- บอกจุดบกพร่องหรือข้อผิดพลาดได้ครบถ้วนกว่า
- บอกค่า technical debt ของโปรแกรมที่ทดสอบได้
- สามารถเปรียบเทียบค่า matrix ตามวันเวลาได้
- เป็น open-source

ข้อเสีย

- UX/UI ใช้งานยาก
- บาง matrix ได้ค่าออกมาไม่ครบถ้วน