TL4 Presentation

CS 383 Software Engineering

What is a Gantt Chart

Oxford Dictionary definition:

- “a chart in which a series of horizontal lines shows the amount of work done or production completed in certain periods of time in relation to the amount planned for those periods.”

Main Goals

- Track Time
- Plan Tasks
- Monitor Future Tasks
- Prioritize tasks

Benton

Why use a Gantt Chart?

Time Management

- Allows you to track out and prioritize certain tasks for yourself and teammates.

Organization

- A clean and organized way for you and your team to rank tasks

Communication

- All teammates have access to the chart allowing for straightforward communication of expectations

Benton

Why use a Gantt Chart? Cont.

Simplicity

- Way to visually see the requirements and priorities of tasks of you and your teammates.

Allocation of Resources ($$$)

- Not only can track hours but can track resource allocation for each task and workers hours
- A good Gantt should be able to track work hours and expected work hours so budgets can be made and met

Benton

When You Should Use a Gantt

Like stated previously Gantt's are used for project management and planning.

You should use a Gantt when:

- Managing Projects
  - Ideally for a linear project where tasks are needed to be done sequentially before moving to the next
- Managing Time of Projects
- Managing Resources of a Project
- Managing Deliverables of a Project

Benton

Gantt Chart Platforms

Choose a desired platform based on requirements. EX:

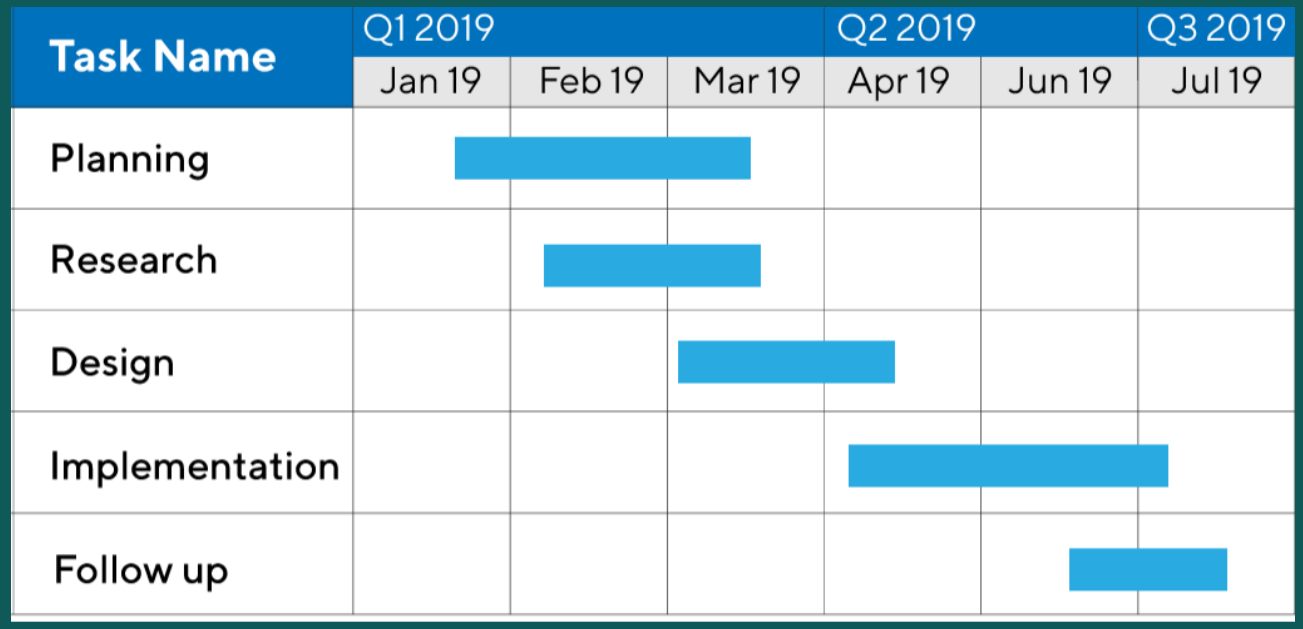
- Excel (Most basic, Difficult to merge to GIT)
- Jira (Online, More dedicated Gantt services and symbols)
- TeamGantt (Online hosted Gantt chart maker)
- Etc.

Things to consider when picking a platform:

- Ease of use (Select a platform that you team will use and understand)
- Where this chart will be hosted.
- Make sure all team members can access and add to the chart
- What you need
- Do you need tasks planned down to the hour, day. Do you need extra plugins?

Benton

Example Gantt Chart



Notice

- Each task is uniquely named for clarity purposes
- Each task has its own timeline which might depend on previous tasks and other requirements
- It's easy to follow and digest when things should be completed and when some should begin.

Benton

How to Add to a Gantt Chart

1. Define the task you are planning out. Note this under the task name section
2. Identify the estimated dedicated work hours to complete said task
3. Note of any prerequisite tasks that might delay the start of the task you are making
4. Place the task on the chart visually
    - If there is a pre task that delays the start of the task you are planning, make sure to account for this when placing on the diagram.

Benton

Using a Gantt Effectively in this Class

When you think about your feature think of everything that would lead up to it being complete.

- Put all requirements as each of their own section on your team's master Gantt.
- Make sure to also outline which tasks are dependent on others and make sure your Gantt follows this structure.

When you complete a task make sure to track this on your Gantt ASAP.

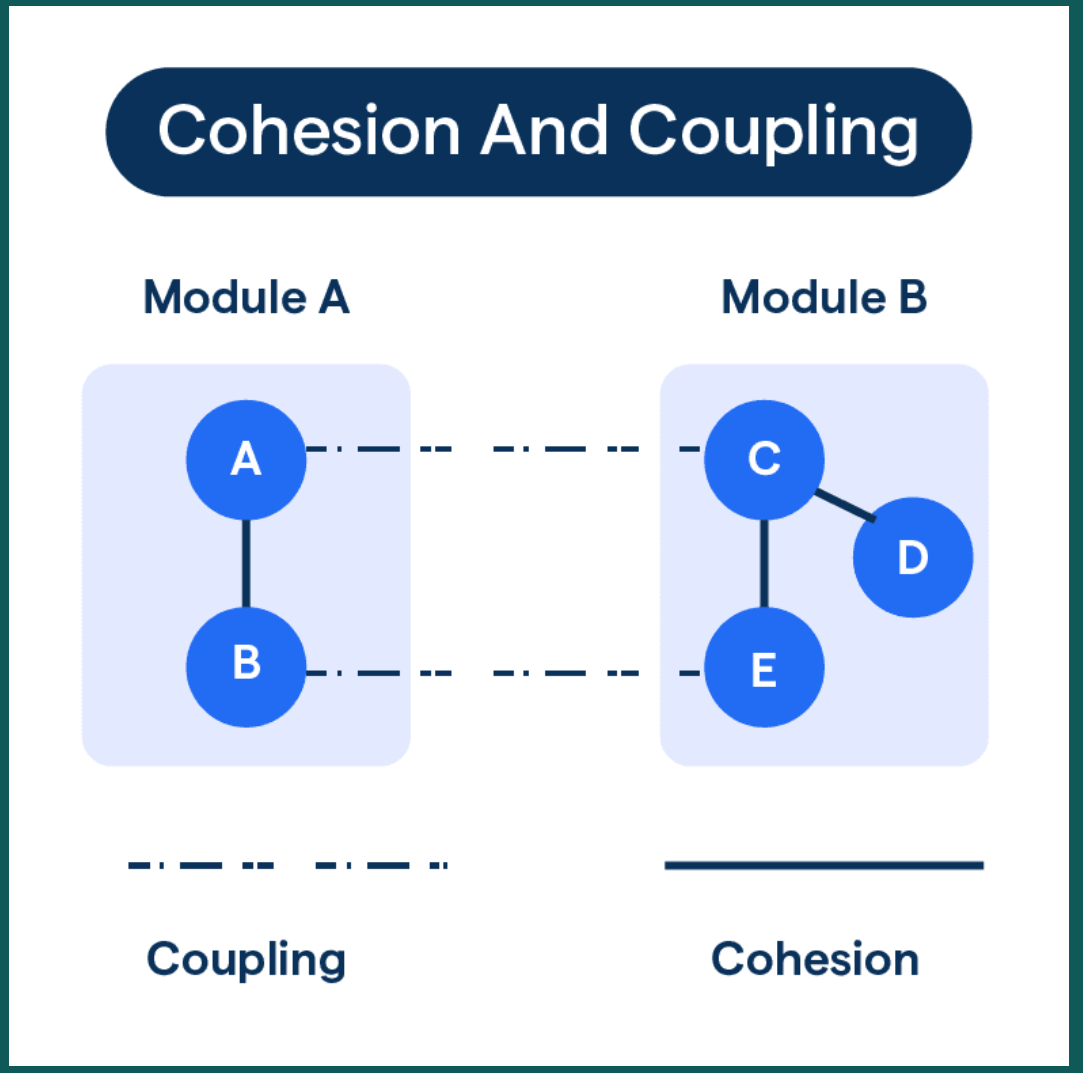
You can even use your Gantt to plan out later tasks to make sure you can dedicate the needed time and resources to complete the full project.

Benton

Coupling and Cohesion

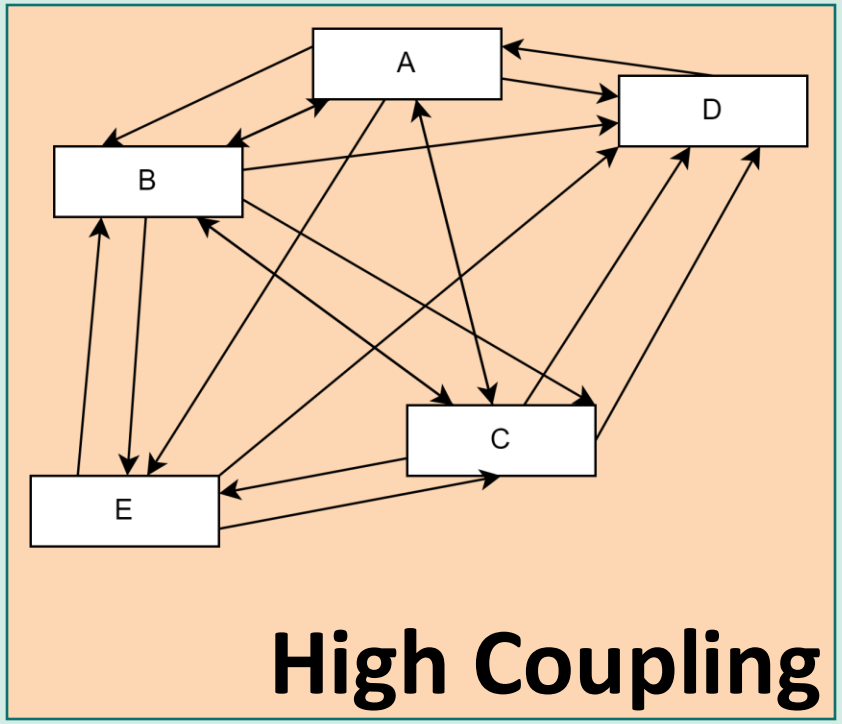
Kate

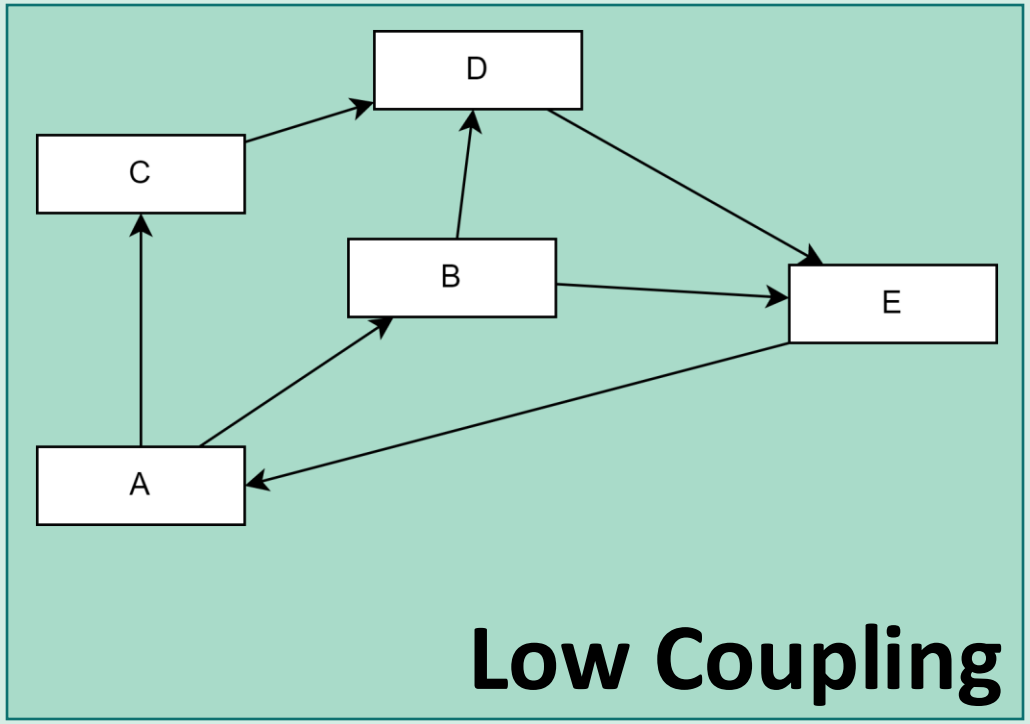
- Metrics to measure how logically sorted a set of functions is
- Lower coupling is better, higher cohesion is better
- WILL BE NEEDED for Postmortem presentation



Coupling

- the measure of the degree of interdependence between the modules



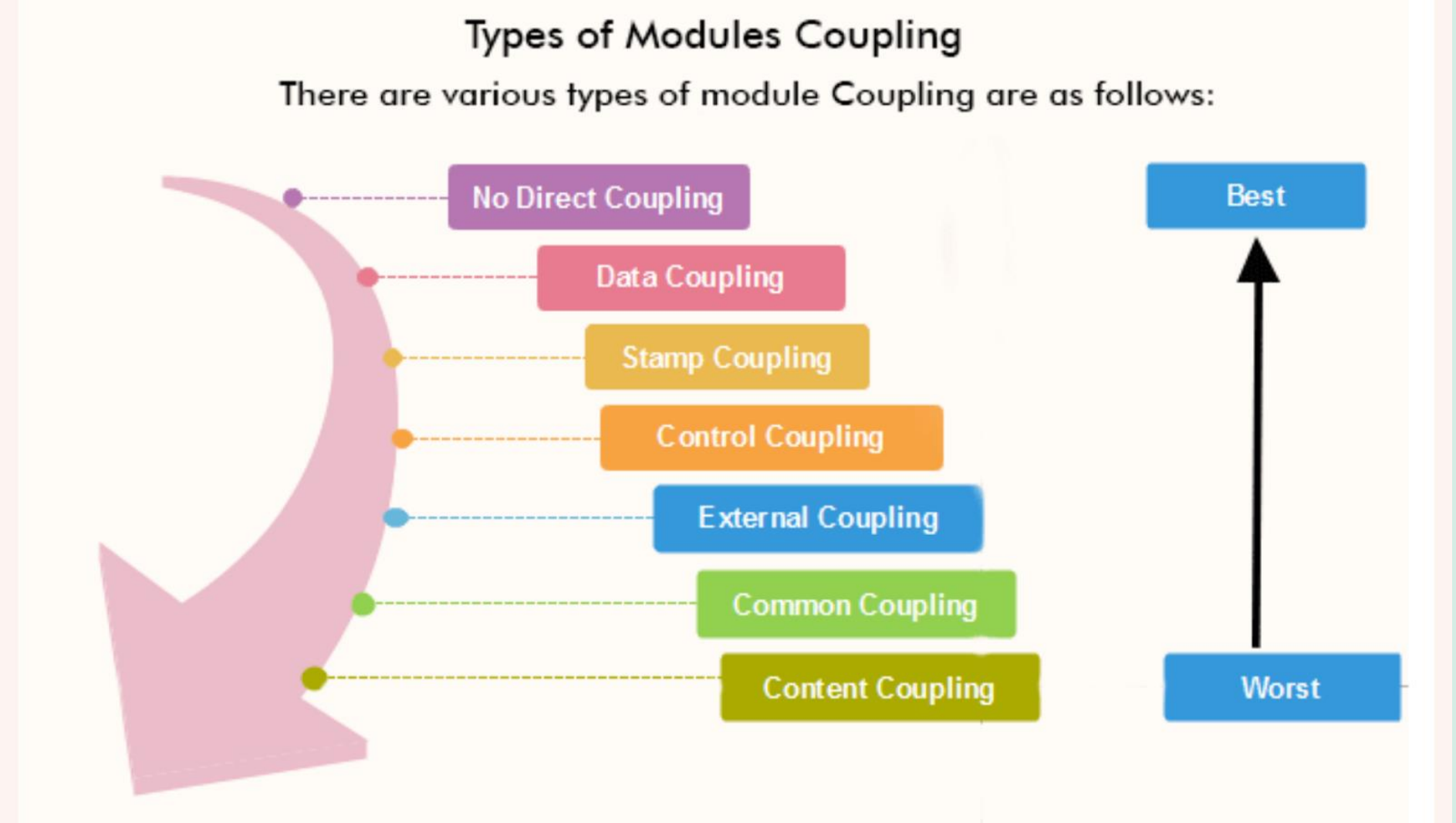


High – strong interconnections where changes affect each other

Low – isolation between modules

Shan

Coupling



Shan

Coupling

No Coupling:

- Modules work w/o direct communication
- Ideal type of connection
- Minimizes impact of changes

Shan

Coupling

1. Data Coupling:

- Modules communicate by passing necessary data
- Components are close to independent of each other
- Loose

Shan

Coupling

2. Stamp Coupling:

- The complete data structure is passed from one module to another as a parameter and use parts
- Changes to a structure can affect modules

3. Control Coupling:

- One module controls another by passing control information
- Ex. Sort function that takes comparison function as an argument

Shan

Coupling

4. External Coupling:

- Modules communicate by exchanging data through external systems
- Risky

5. Common Coupling:

- Modules share same global data and resources
  - Used and modified by different modules
- Disadvantage: difficulty in reusing

Coupling

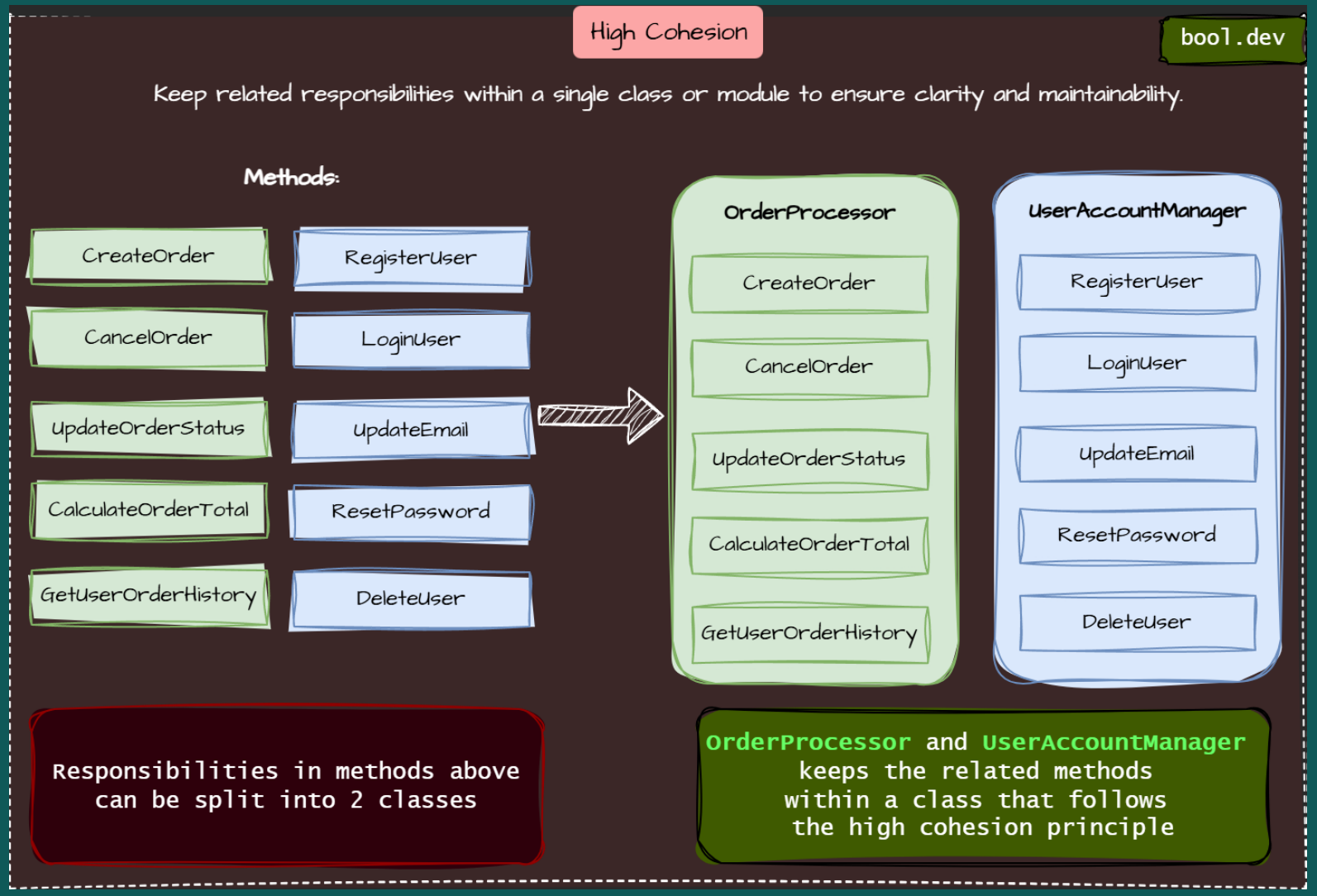
6. Content Coupling:

- One module relies on or can modify the internal details of another module
- Strongest coupling
- Worst form, should be avoided due to high dependencies

Cohesion

Kate

- Degree to which elements in a module belong together



Cohesion

Kate

- What:
  - Functionalities share significant similarities
  - Functions carry out a small number of related activities
  - Related functions are in the same source file or otherwise grouped together
- Why:
  - Improved maintainability and reuse of components

Cohesion

Kate

- Coincidental Cohesion
  - Functions share nothing in common besides being near each other
- Logical Cohesion
  - Functions perform similar operations logically, despite having different underlying operations
    - Ex: Mouse and keyboard input in an InputHandler

Cohesion

Kate

- Temporal Cohesion
  - Functions are called at the same or closely grouped times
- Procedural Cohesion
  - Functions are called one after another in a fixed sequence
- Informational Cohesion
  - Functions operate on the same data

Cohesion

Kate

- Sequential Cohesion
  - Output from one function is chained to input in the next
  - Related to Procedural Cohesion
- Functional Cohesion
  - Functions all contribute to a single well-defined task handled by the entire module

GRASP

General
Responsibility
Assignment
Software
Patterns

GRASP Patterns Description

Each GRASP pattern comes with:

- Discussion
- Contradiction
- Benefits
- Related Patterns or principles

Some key notes about these GRASP patterns:

- These patterns provide advice on **assignment** and **responsibilities**
- Patterns may conflict each other in advice
  - So we must use best judgment when applying certain/multiple patterns
  - The extra sections will help us with this
- This work is not mechanical!

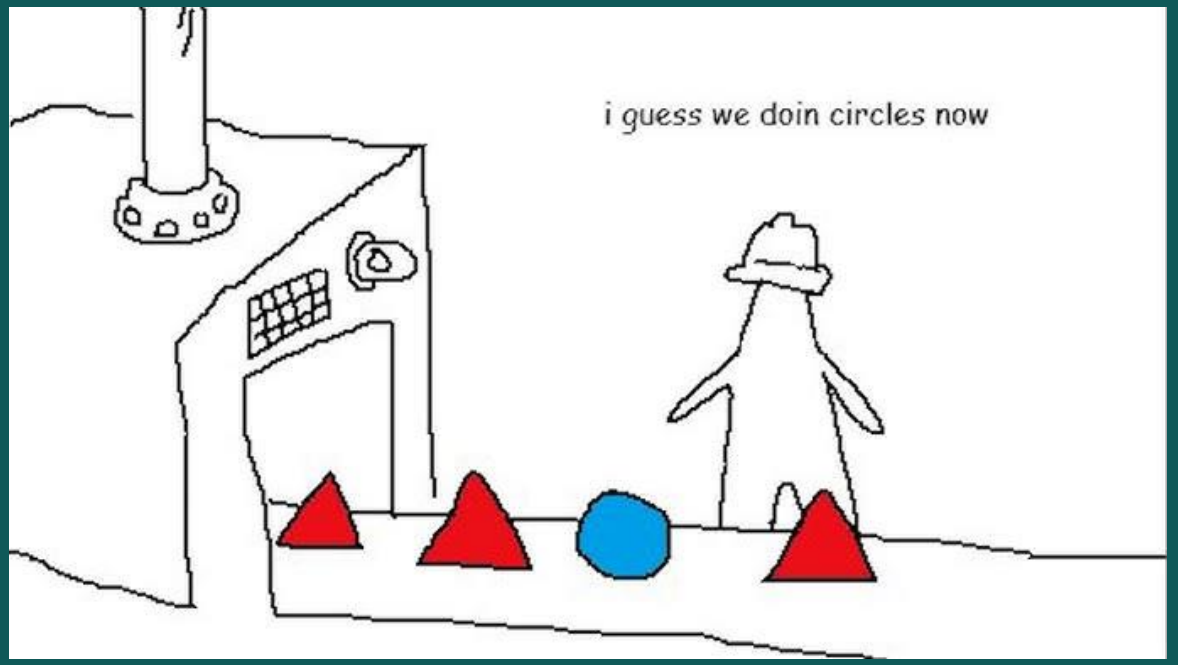
GRASP Pattern - Information Expert

Zach

"Assign a responsibility to the class that has the necessary information to fulfill it, promoting high cohesion and minimizing coupling"

- High Coupling – Interdependence
- High Cohesion – Independence

The Information Expert pattern avoids the issue within the triangle factory.



GRASP Information Expert

Zach

Question – What is a General principle of assigning responsibilities to objects?

Solution – Assign responsibilities to an "Expert Class" which is a class with the information to complete the task.

Don't assign circle creation to the TriangleFactory class.

GRASP Information Expert

Zach

Benefits

- Encapsulates data well
- Class behavior is spread across smaller lightweight classes

Detriments

- *Can* lower cohesion by adding responsibility to a class
- *Can* raise coupling if the added responsibility requires communication with a separate module

Varying levels of coupling and cohesion aren't inherently bad, but consider how it should be used given your context

GRASP Information Expert

Zach

How to implement

- Use private methods and variables
- Reduce data sharing between different modules (when applicable)
- Approach coding with these principles in mind;
  - With a task in mind, will this lower coupling, or raise it?
  - It's much easier to start with principles in mind than to fix conflicts

GRASP Information Expert

Zach

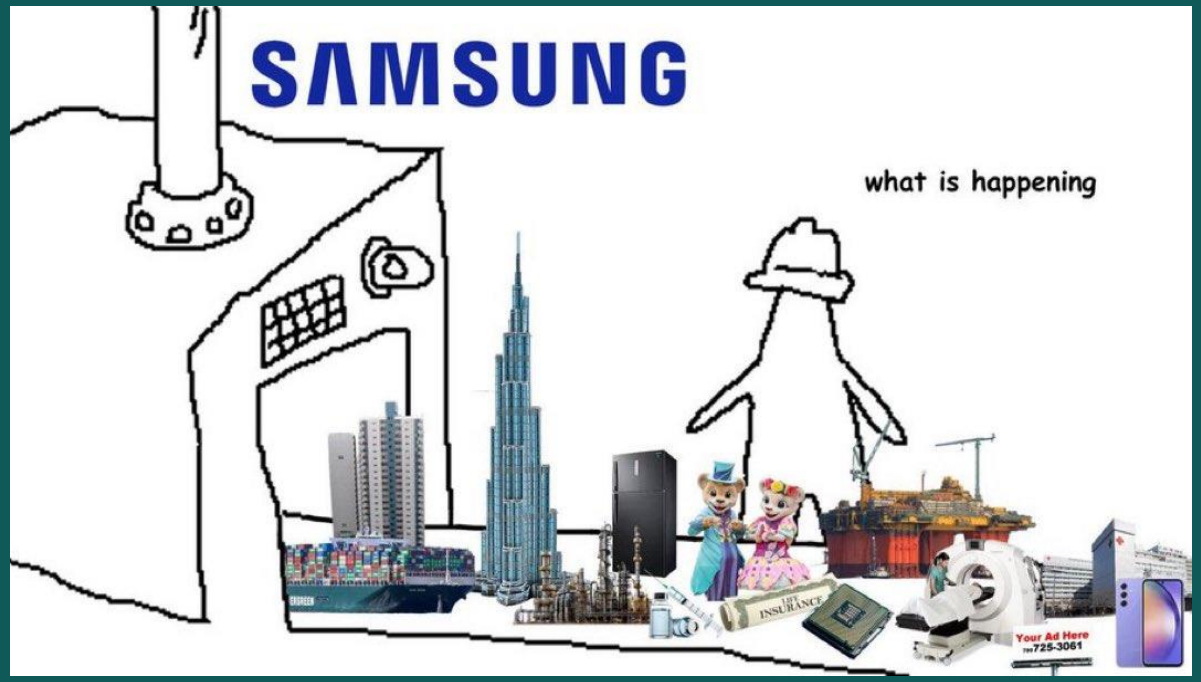
```
// ❌ Factory has too much responsibility
class ShapeFactory {
    public static Object createShape(String type, double... dimensions) {
        if (type.equalsIgnoreCase("triangle")) {
            return new Triangle(dimensions[0], dimensions[1]);
        } else if (type.equalsIgnoreCase("circle")) {
            return new Circle(dimensions[0]);
        } else {
            throw new IllegalArgumentException("Unknown shape type");
        }
    }
}
```

```
// ✅ The factory now delegates responsibility to the expert classes
class ShapeFactory {
    public static Shape createShape(String type, double... dimensions) {
        switch (type.toLowerCase()) {
            case "triangle":
                return Triangle.create(dimensions[0], dimensions[1]);
            case "circle":
                return Circle.create(dimensions[0]);
            default:
                throw new IllegalArgumentException("Unknown shape type");
        }
    }
}
```

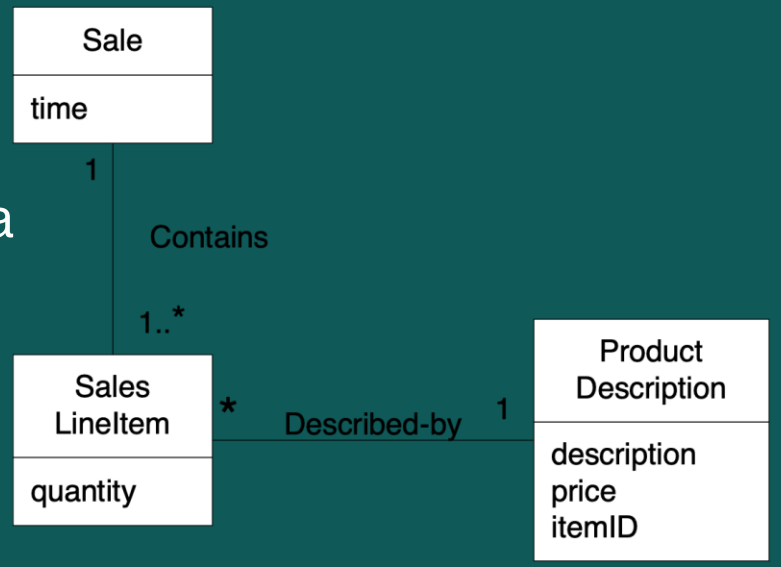
GRASP Information Expert

Zach

Don’t put your group through this:



GRASP Pattern – The Creator



Problem "The Creator" Addresses: Who is able/allowed to create a new instance of some class.

Example: Who's responsible for creating "SalesLineItem" class?

In OO programming the creation of objects is the one of the most common activities

So the need for "The Creator" is evident. The specific aims of the creator are to achieve:

- Low coupling
- Increased clarity
- Increased encapsulation
- Increased reusability

Benton

GRASP Creator

Problem:

- Who's responsible for creating a new instance of some class?

Solution:

- Assign class B the responsibility to create an instance of class A if **AT LEAST ONE** of these is true:
  - B "contains" or compositely aggregates A.
  - B records A
  - B closely uses A
  - B has initializing data for A (*i.e.*, B is and *Expert* with respect to creating A)

Benton

GRASP Creator

Discussion:

- Intent is to support **low coupling** (i.e. creator is found that already needs to be connect to created object)
- Initializing data is sometime indicator of a good Creator
  - Some object constructors have complex signatures
  - Which objects have the information needed to supply parameter values for such constructors?

Contradictions:

- Creation can be complex (recycled instances, creating an instance from a family of similar classes)
- May wish to use Factory Method or Abstract Factory instead in these cases

GRASP Creator

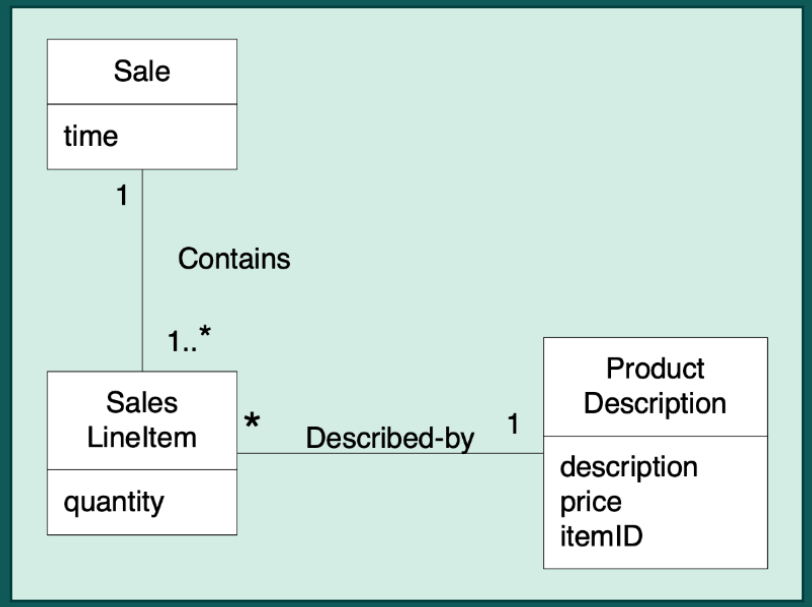
Benefits:

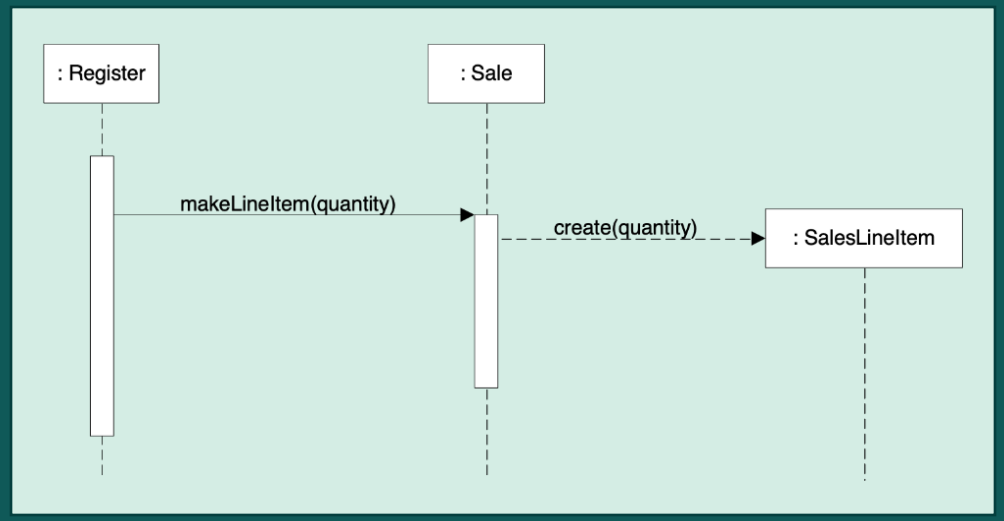
- Low coupling is supported, which implies lower maintenance dependencies and higher opportunities for reuse
- Why is this good?

Related Patterns or Principles:

- Low coupling
- Factory Method and Abstract Family
- Whole-Part pattern: defines aggregate objects that support encapsulation of components

GRASP Creator





Sale will contain many SalesLineItem objects

Creator GRASP pattern suggests Sale is one object that could fulfill this responsibility

Consequences:

MakeLineItem becomes a method in Sale. We capture this decision in our UML model diagrams.

Benton

GRASP Creator

Source: https://www.sourcecodeexamples.net/2018/06/creator-grasp-pattern.html

```
class Sale {
    List<SalesLineItem> salesLineItem
    = new ArrayList<SalesLineItem>();
    //...
    public void addLineItem(ProductSpecification prodSpec,int quantity) {
        salesLineItem.add(new SalesLineItem(prodSpec, quantity);
 return salesLineItem;
    }
}
```

As you can see the addLineItem has now become a method in the Sales class. This means that the Sale class has now become the creator of the addLineItem method and is able to call and create objects whenever.

Benton

GRASP Pattern - Polymorphism

- What is Polymorphism in GRASP?
  - A principle that allows objects to be treated as instances of their parent class rather than their actual class.
  - Helps in designing systems where behavior varies depending on object type without modifying existing code.
  - Encourages loose coupling and scalability.

Jeffrey

GRASP Polymorphism

- Problem Addressed by Polymorphism
  - How do we handle different behaviors for related types without conditional logic?
  - Avoiding large `if-else` or `switch` statements when handling multiple object types.

- Solution
  - Define a common interface or abstract class for related behaviors
  - Allow subclasses to override and implement specific behavior
  - Delegate responsibility to the correct subclass dynamically

Jeffrey

GRASP Polymorphism

- An Example

- The `Checkout` class does not need to check which payment method is used; it simply calls `processPayment`

```java
interface PaymentMethod {
    void processPayment(double amount);
}

class CreditCardPayment implements PaymentMethod {
    public void processPayment(double amount) {
        System.out.println("Processing credit card payment of $" + amount);
    }
}

class PayPalPayment implements PaymentMethod {
    public void processPayment(double amount) {
        System.out.println("Processing PayPal payment of $" + amount);
    }
}

class Checkout {
    void completeTransaction(PaymentMethod method, double amount) {
        method.processPayment(amount);
    }
}
```

Jeffrey

GRASP Polymorphism

- Discussion
  - Using polymorphism enables designing flexible and scalable systems.
  - It reduces redundancy and simplifies code maintenance by centralizing behavior in a common superclass or interface.
  - Ensures that new behaviors can be added without modifying existing code, supporting the Open/Closed Principle.

However

- Contradictions
  - It can introduce unnecessary complexity if overused
  - May lead to improper class hierarchies where behavior does not logically belong to subclasses.
  - Overuse can make debugging difficult due to increased abstraction layers

Jeffrey

GRASP Polymorphism

- Benefits of Polymorphism in GRASP
  - Reduces coupling between classes.
  - Simplifies code maintenance and extensions.
  - Encourages modularity and reusability.
  - Enhances code flexibility by allowing objects to be used interchangeably.
  - Supports design principles such as Open/Closed Principle and Dependency Inversion.

Jeffrey

GRASP Polymorphism

- Related Patterns
  - *Indirection: Delegates responsibility to an intermediary*
  - *Protected Variations:* Uses stable interfaces to shield against change
  - *Information Expert:* Assigns behavior to the most knowledgeable class

Jeffrey

GRASP Pattern - Protected Variations

- **Problem:** How do we create our system in a way where we can change something in one place and still retain functionality of the system?

- **Solution:** Organize responsibilities around stable interfaces

Shan

GRASP Protected Variations

Interfaces

- Type of abstract contract that describes what a class can do
  - Actual implementation is left to class
  - Ex. Must have a GetData() function
    - Separate points of instability (susceptible to change) from rest of code

Benefits

- Isolates changes, making the system more robust and easier to modify
- Good for code that changes often
- Low coupling

Shan

GRASP Protected Variations

Discussion:

- Intended to design the system to *protect* against *variations* or changes that may affect other parts of the code

Contradictions:

- Over-Abstraction – too complex, low readability
- Abstractions could become unstable themselves

Shan

GRASP Protected Variations

Related Patterns

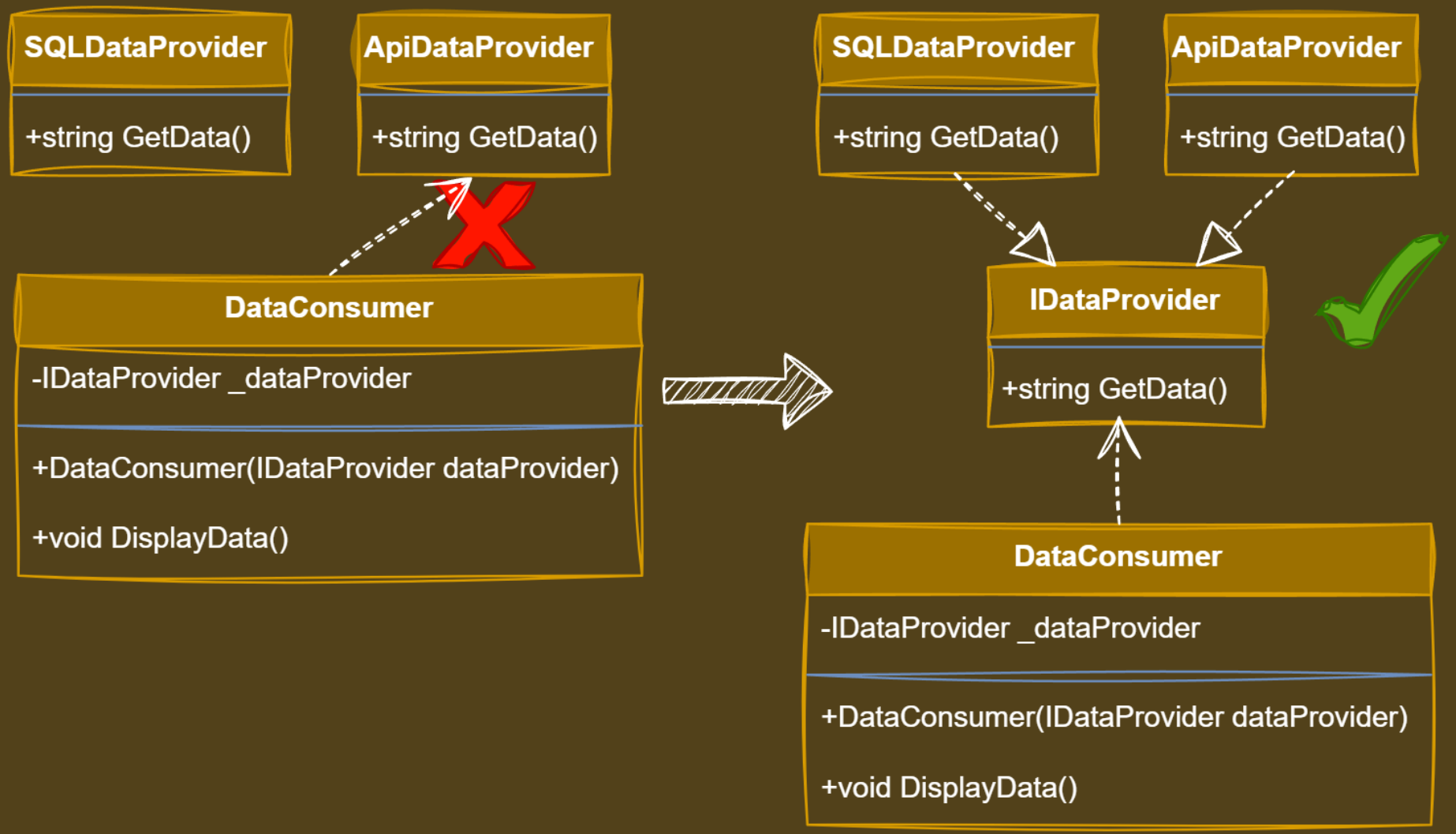
- Low coupling – minimize dependencies between objects
- Open/Closed Principle from SOLID
  - Software entities should be open for extension but closed for modification

Shan

Protected Variations

bool.dev

Design the system to protect against variations or changes that may affect other parts of the code.



Shan

GRASP Pattern – Controller

Kayra

- **Problem:**
  - What should be responsible for handling an input system event?
- **Solution:**
  - Assign responsibility to a class based on:
    - Class is "the root" object that represents the overall system
    - A new class responsible for handling a specific user interaction

GRASP Controller

Kayra

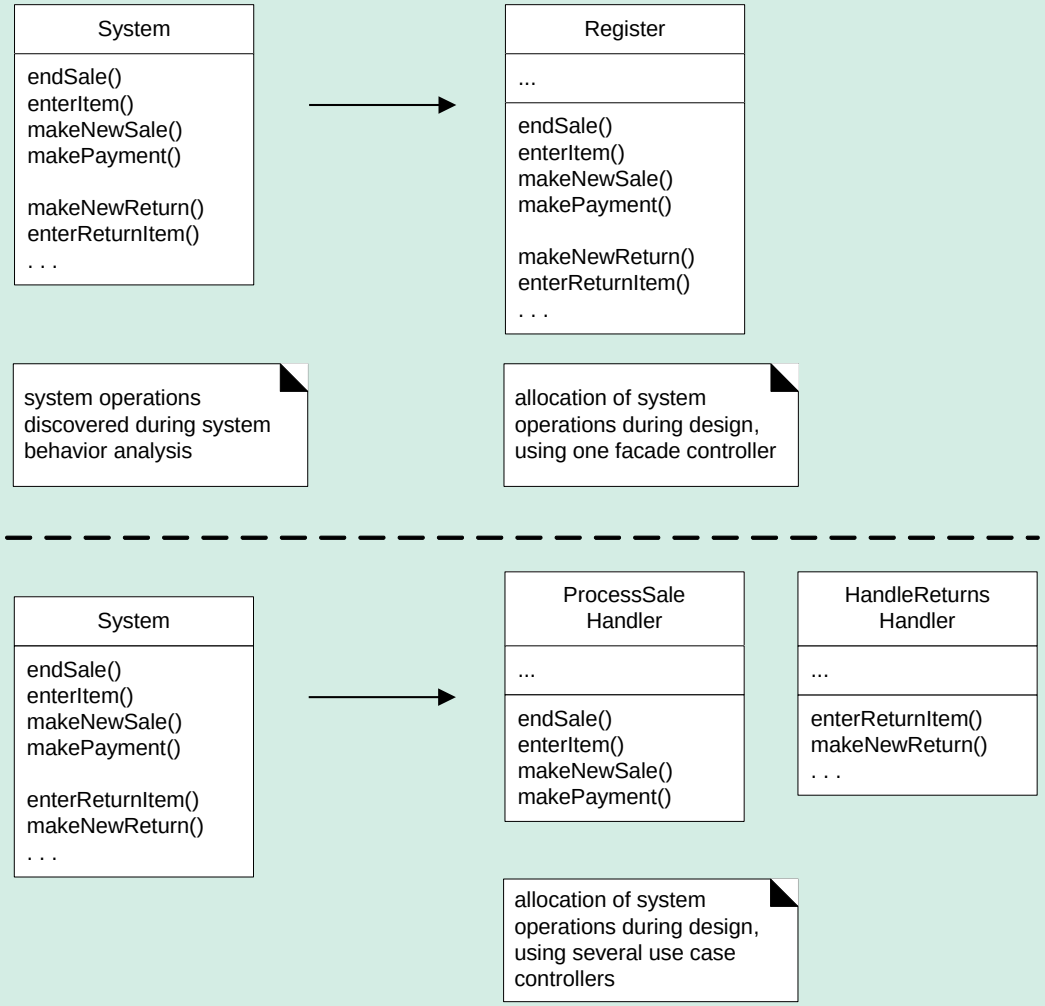
- A controller object is a non-user interface object responsible for receiving and handling a system event.

- **Benefits**
  - Separation of concerns
  - Improves reusability, maintainability, and scalability
  - Supports testability – does not require user interaction

GRASP Controller

Kayra

- **How the controller works**
  - The UI triggers a system event
    - sent by a button click or a request
  - A controller object is responsible for handling that event
  - The controller delegates tasks to domain objects
  - The controller returns a response to the UI



GRASP Controller

Kayra

- **Discussion**
  - Intended to assign the responsibility of handling a user input to a non-UI class and act as an intermediary between the UI and the business logic
- **Contradictions**
  - Increased complexity
  - Tight coupling: when two or more classes/components are highly dependent on each other

GRASP Controller

Kayra

- **Related Patterns**
  - Information expert – ensures correct delegation
  - Low coupling – keeps the system modular and maintainable

GRASP Pattern – Pure Fabrication

Sikha

- Problem: How do we assign responsibilities when no natural class seems to be the right choice?

- Solution: Create a Pure Fabrication class, a class that does not represent a real-world entity but is introduced to handle a specific responsibility. This helps achieve low coupling, high cohesion, and separation of concerns.

GRASP Pure Fabrication

Discussion:

- The pattern is used to encapsulate behaviors that don’t belong to any domain object.
- It helps separate technical concerns from the business logic of the system.
- A fabricated class may perform tasks like logging, encryption, or persistence.

Contradictions:

- Overuse can lead to too many artificial classes, increasing complexity.
- Might introduce unnecessary layers, making debugging harder.

GRASP Pure Fabrication

Benefits:

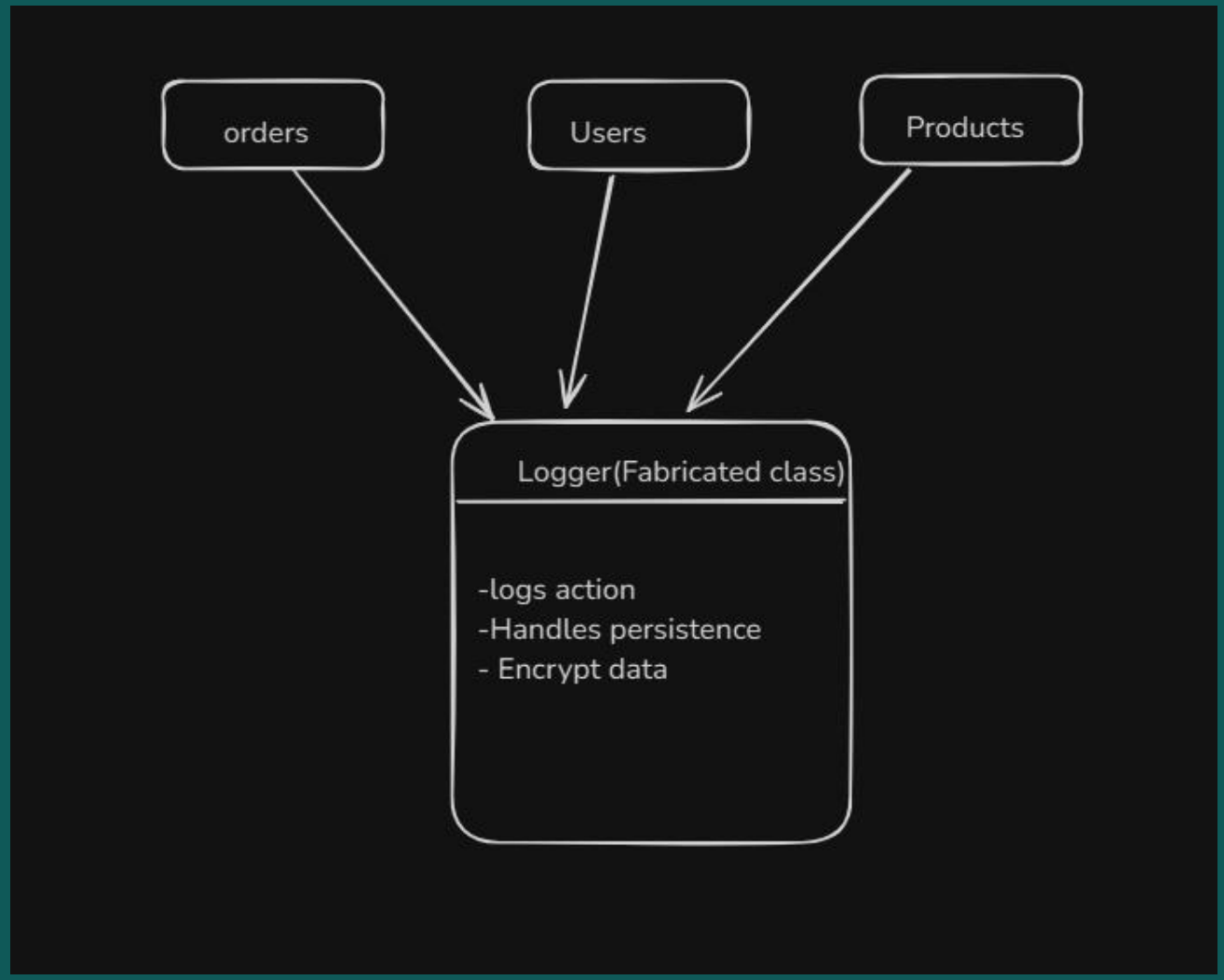
- Reduces coupling by keeping domain objects focused on their core responsibilities.
- Increases cohesion by grouping related behaviors into separate, manageable classes.
- Enhances reusability, making it easier to use the fabricated class across different parts of the system.

GRASP Pure Fabrication

Related Patterns:

- Indirection – Delegating responsibility to a mediator to lower coupling.
- Low Coupling – Minimizing dependencies between objects.
- Facade Pattern – Providing a unified interface to a set of interfaces in a subsystem.

GRASP Pure Fabrication



GRASP Pattern - Indirection

Problem:

- Where are responsibilities assigned if we want to avoid direct coupling between two or more things?

Solution:

- Add an intermediary object with responsibility to mediate between two or more components.

GRASP Indirection

Discussion

- Adding intermediaries gives code a "middle-man" between two programs.
- Organizes details and variations into a common abstraction.

Contradiction

- Adding intermediaries can affect the memory and performance usage of a program.

GRASP Indirection

Benefits

- Improved reusability of code.
- Code is easily maintainable and adaptable.
- Changes can be made without affecting an entire system.

GRASP Indirection

Related Patterns:

- Low coupling: reducing interconnectedness of objects.
- Polymorphism: encouraging scalability.

GRASP Pattern - High Cohesion

- Problem:
  - Which functionality should be placed in which classes?
- Solution:
  - Closely related functions should be placed in the same class
  - Unrelated functions should be fragmented into different classes

Kate

GRASP High Cohesion

- Benefits:
  - Easy to locate specific functions if all functions are well-sorted into classes
  - Classes do not need to rely too heavily on internal specifics of other classes, making modifications or replacements easier
- Drawbacks:
  - The same function might be related to multiple different sets of functionality

Kate

GRASP High Cohesion

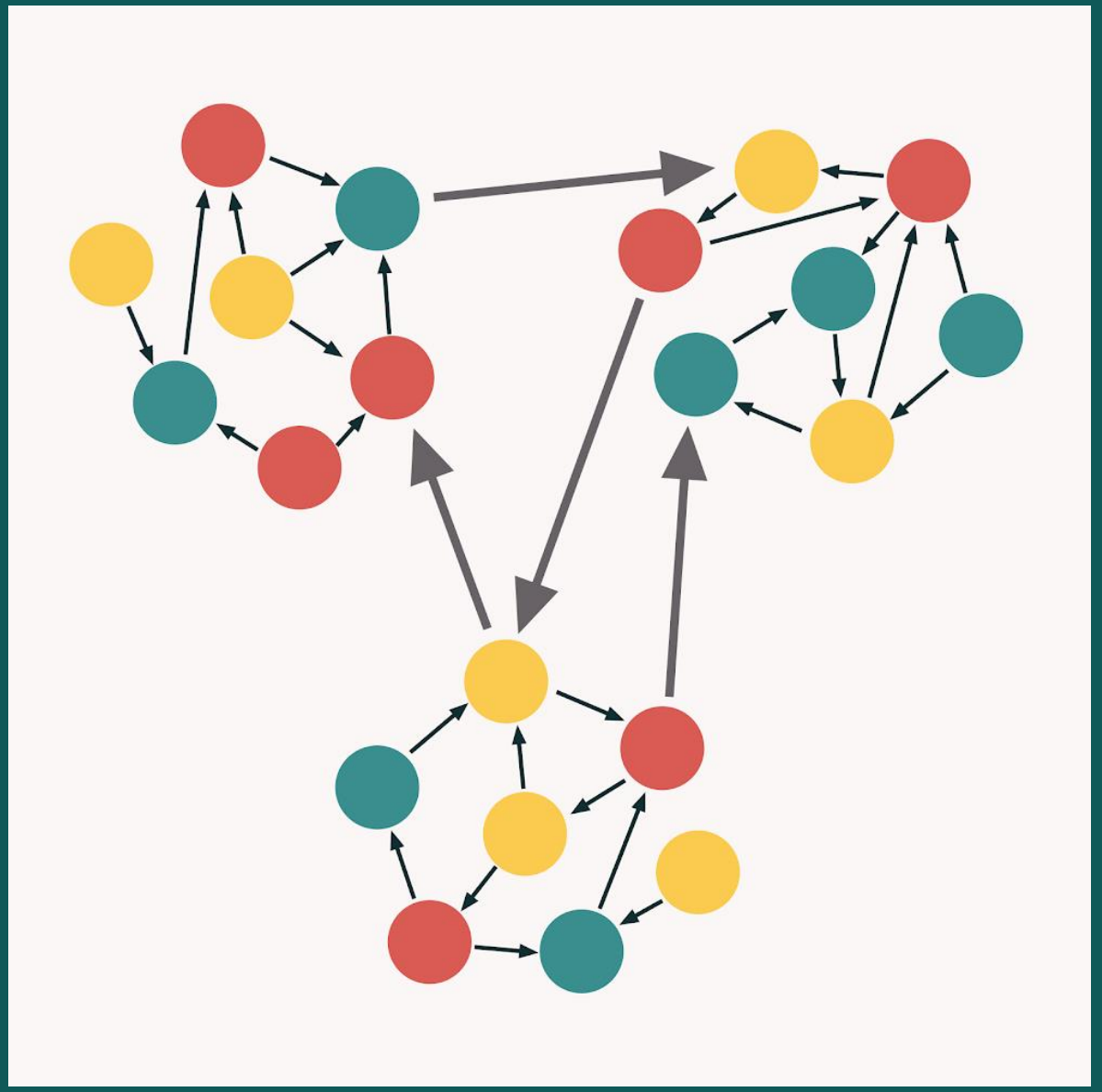
- Exceptions
  - We might want shared umbrellas for functions which are individually too small to warrant an entire class.
  - In those cases, an XYZHelper class which contains a large number of low-cohesion miscellany which can't be fit anywhere else is acceptable.
- Related Patterns
  - Cohesion and coupling are the same phenomenon, but within and between classes respectively

Kate

Kate

GRASP High Cohesion

- Example
  - Modules depend primarily on their own subcomponents
  - Dependence on internals of other modules is minimized



GRASP Pattern – Low Coupling

- **Problem:** How do we handle high dependency and increased reuse of our code?

- **Solution:** Minimize dependencies between classes to reduce the impact of changes.

Shan

GRASP Low Coupling

Discussion

- As few connections as possible between our classes, modules, files, etc.

Benefits

- Code easier to update and maintain
- Efficient, reusable, understandable
- The fewer connections, the more stable the system

Shan

GRASP Low Coupling

Contradictions

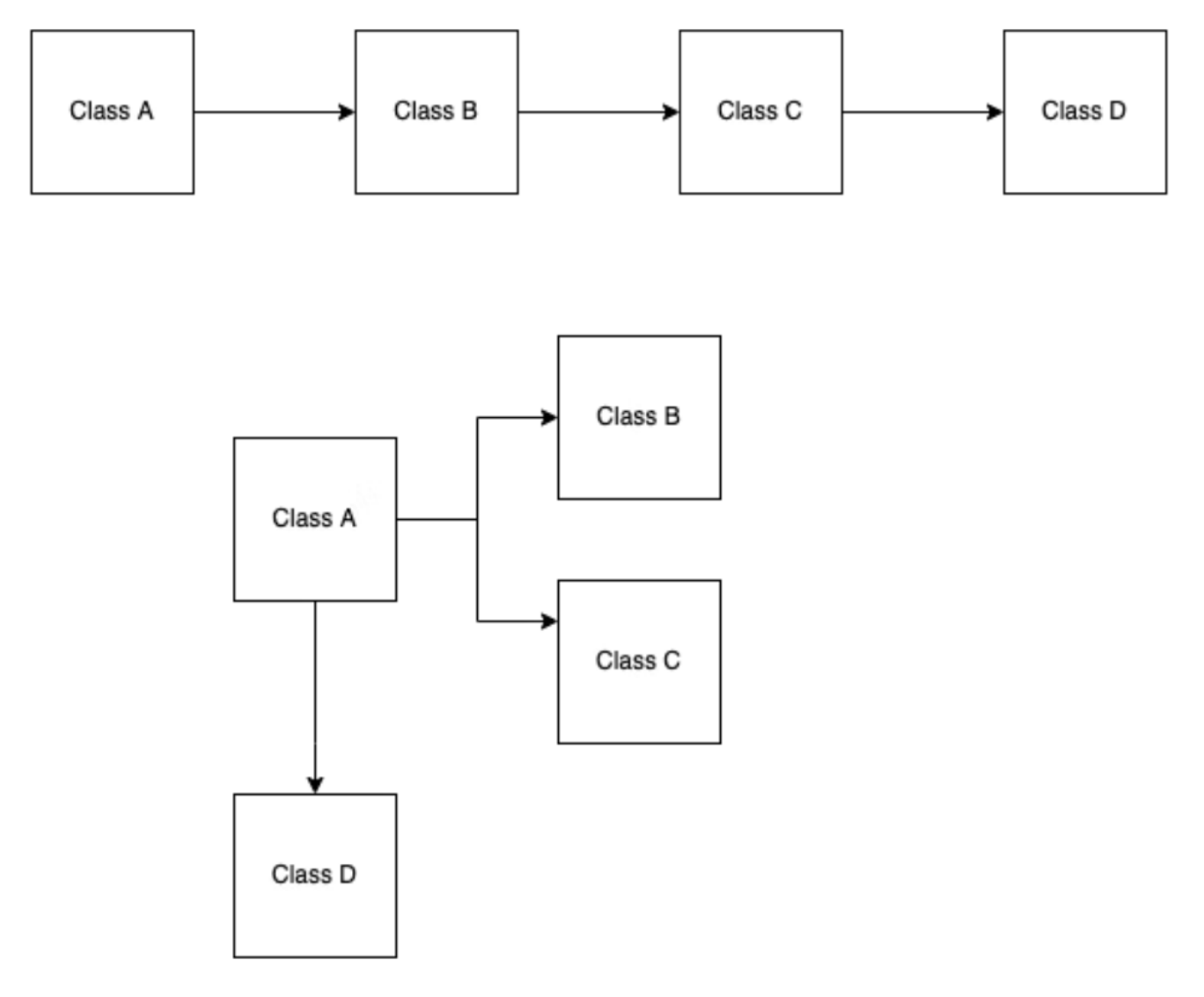
- Some dependencies are unavoidable - calling a function
- Avoid connection if not needed

Related Patterns

- Related to all other patterns
- Creator – will only have connections where the created object will be used (where required)

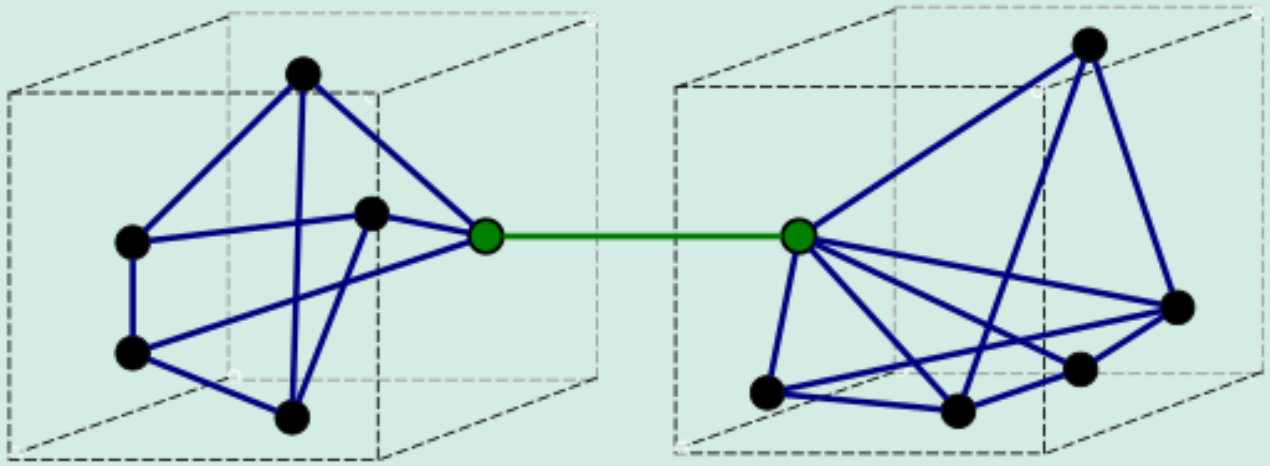
Shan

GRASP Low Coupling

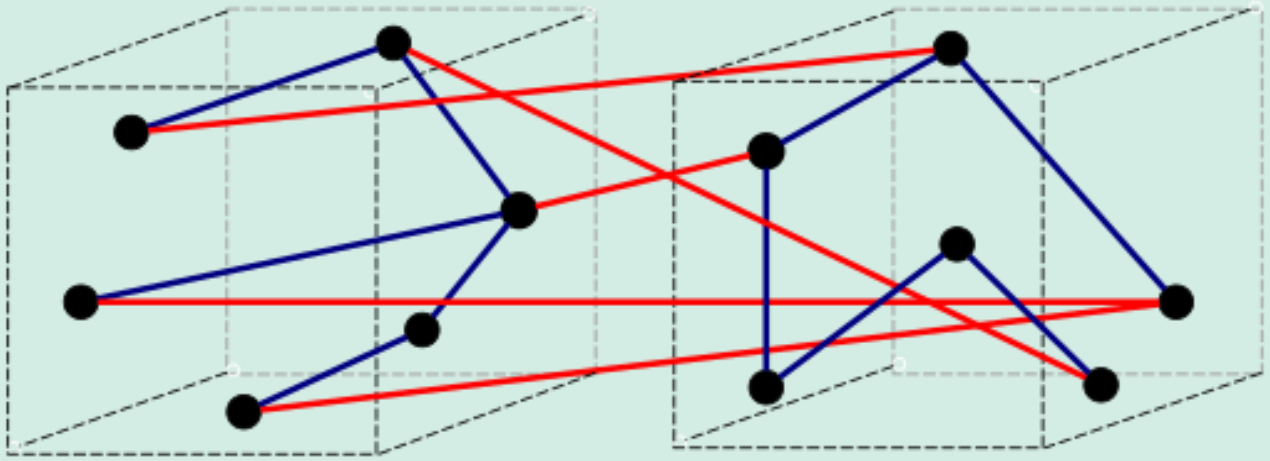


Shan

GRASP Low Coupling



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

Shan

Gantt Roast – Kirklind Signature

Conner				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Downloading Unity Environment	2	3	Complete																
Drafting RFP, Champion, Sequence, Class	2	6	Complete																
Working on TL Platformer	5	7	Complete																
Adding Stress test	Office on the web Frame	8	Complete																
Adding Boundary Tests	3	5	Complete																
Adding Main Menu (Basic)	1	1	Complete																
Adding Pause Menu (Basic)	1	1	Complete																
Adding Game Over (Basic)	1		this week																
Adding Help Menu (Basic)	3		Planned																
total																			
Aaron				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Downloading Unity Environment	1	2	Complete																
Drafting RFP, Champion, Sequence, Class	6	7	Complete																
Working on TL Platformer	8	9	Complete																
Adding Visual Hearts (Basic)	2	2	Complete																
total																			

It looks like Aaron might be falling behind if we were looking just at the Gantt chart, compared to others.

Divyansh				1	2	3	4	5	6	7	8	9	10	11	12
Downloading Unity Environment	1	2	Complete												
Working on TL Platformer	2	2	Complete												
Drafting RFP, Champion, Sequence, Class	5	7	Complete												
Animator Controller & Animation States	2	3	Complete												
Implement Animation Transitions & Logic	3	3	Complete												
Testing & Debugging Animation Issues	2	3	Complete												
Asset Loader	3		this week												
total	18	20													
		320													

Seems as though this team member does not have any features planned for the future. Is this a correct assumption?

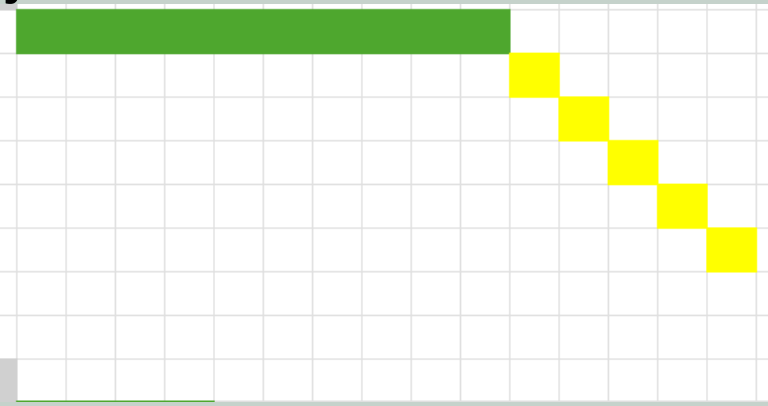
Gantt Roast – Green Sky Games

Shan	54	Requirements Collection	Done
	55	Make sprites for items	In Progress
	56	Program effects for Power-Ups + PoisonApple	In Progress
	57	Program all Item reactions to Player collision – ite	In Progress
	58	Program result of Player clicking Use Now and Pla	In Progress
	59	Program disappearance of Items when used/disc	In Progress
	60	Program Initial Inventory System and subclasses	In Progress
	61	Program how/when Inventory is displayed	Planned
	62	Program placeholder for text shown when item is	Planned
	63	Program functionality of adding items to Inventory	Planned
	64	Program effect of Player trying to add too many ite	Planned
	65	Program automatic addition to Inventory for Colle	Planned
	66	Program increase in Inventory slots when clueCou	Planned
	67	Program result of Player clicking Use Now on Gold	Planned
	68	Program result of Player clicking Use on Golden A	Planned
	69	Program OwlsWing effect when Player clicks Use	Planned
	70	Program CanOfTuna effect when Player clicks Us	Planned
	71	Testing	Planned

Liz	73	Requirements	Done
	74	Creating Visual Main Menu Screen	Done
	75	Coding Main Menu behavior scripts	Done
	76	Creating Pause menu Screen	In Progress
	77	Coding Pause menu behavior and scripts	Done
	78	Creating inventory button and menu	Planned
	79	Coding Inventory button and behavior script	Planned
	80	Designing Victory Screen	Planned
	81	Coding Victory Screen behavior scripts	Planned
	82	Designing Game Over Screen	Planned
	83	Coding Game Over screen behavior scripts	Planned
	84	Testing	Planned
	85	**Total**	

Wouldn't there be low cohesion if Shan is doing inventory instead of Liz?

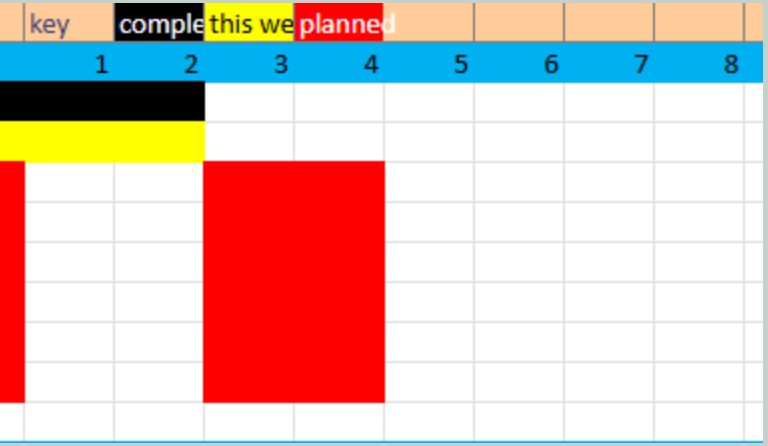
Mark	45	Basic Character Customization	Done	6	10
	46	Expand Character Customiza on (Track Combat S	In Progress	6	1
	47	Enable Basic Character Progression (Gaining XP,	In Progress	16	1
	48	Expand Character Progression (Track Power-Ups,	In Progress	6	1
	49	Implement NPC tracking	In Progress	10	1
	50	Finalize Special Cases (Respawning when Health	In Progress	6	1
	51	Finalize Tracking Stats	Planned	6	
	52	Testing	Planned	2	
	53	**Total**		58	15



Looks like Mark has very few tasks listed. Will this be an issue moving forward?

Gantt Roast - defaultCompany

		predicted time(hrs)	time spent(hrs)	Status	key	comple	this we	planned
2	Monse							
3	Repo + Github Setup	2	2	complete				
4	Map Design	6	2	this week				
5	Database Construction	2	0	planned				
6	User Documentation	6	0	planned				
7	Programming	5	0	planned				
8	Artwork	4	0	planned				
9	Testing	3	0	planned				
10	Installation	1	0	planned				
11	totals	27	2					



Improper layout, lower time use than other team members. Might be falling behind?

19	Justin			
20	Requirements Collection	5	2	complete
21	Target Search System	10	2	complete
22	Basic NPC Movement	10	2	complete
23	Minimap	5	1	complete
24	Minimap icons	5	0	this week
25	More Complex AI Behaviors	10	0	planned
26	totals	45	7	

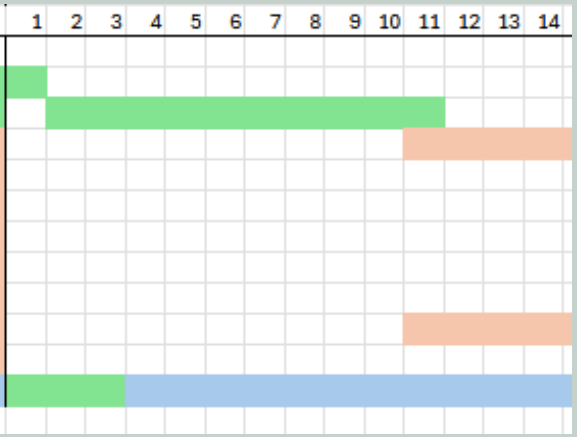
Suspiciously low time uses for everything to be complete. MVP is not the same as being finalized.

If data is accurate, will Justin have enough tasks left for the remainder of the semester?

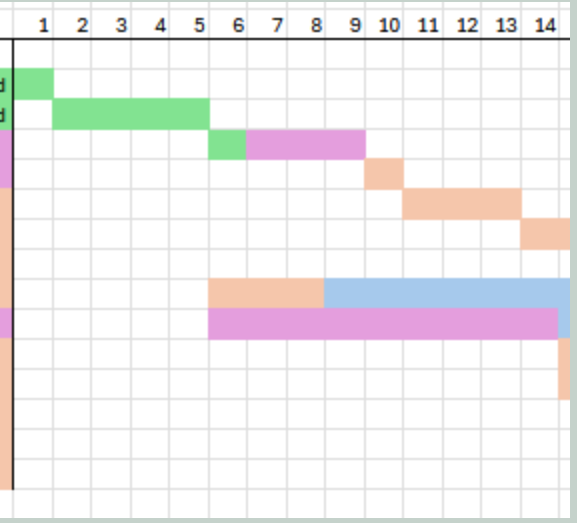
Gantt Roast - Glorbos

Why are these individual hours low? Perhaps behind on updating?

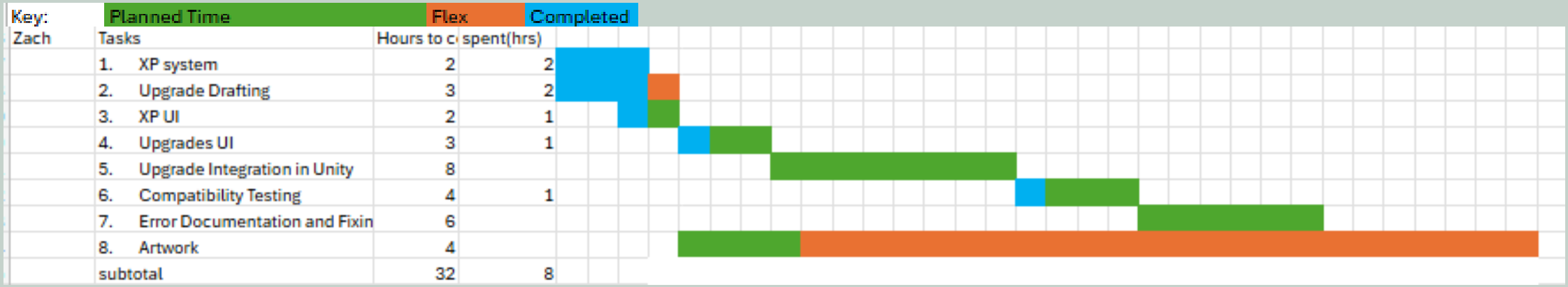
	Tristan (Team Lead 3)	Predicted Time (Hrs)	Time Spent (Hrs)	% Complete	Status
	Setting up Unity	1	1	100%	Completed
1	Requisite Documents	6	10	100%	Completed
2	Enemy Drop System	6	3	50%	Planned
3	Player Pickups System	4		0%	Planned
4	Temporary Powerups	8		0%	Planned
5	Upgradeable Attributes	4		0%	Planned
6	Permanent Upgrades	8		0%	Planned
7	Comedic Upgrades	8		0%	Planned
8	Debugging	40		0%	Planned
9	Automated Testing	10		0%	Planned
10	Documentation	50	3	0%	Slack
		95	3		



	Tavian (Team Lead 4)	Predicted Time (Hrs)	Time Spent (Hrs)	% Complete	Status
	Setting up Unity	1	1	100%	Completed
1	Requisite Documents	4	4	100%	Completed
2	UI Framework	4	1	20%	This Week
3	UI Testing	1		0%	This Week
4	Start Menu Design	3		0%	Planned
5	Pause Menu Design	3		0%	Planned
6	Help Menu Design	3		0%	Planned
7	HUD Design	3		0%	Planned
8	Finding SFX/Music	9		0%	This Week
9	Add SFX to Actions	4		0%	Planned
10	Add Music to Levels & Fights	3		0%	Planned
11	Researching VFX	2		0%	Planned
12	Designing VFX	11		0%	Planned
13	Final Testing	1		0%	Planned
		52	6		

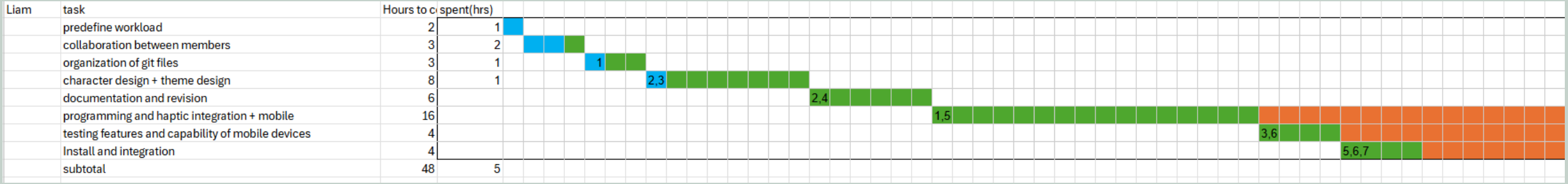


Gantt Roast – SuperNova Games



Key:	Planned Time	Flex	Completed
Zach	Tasks	Hours to c	spent(hrs)
	1. XP system	2	2
	2. Upgrade Drafting	3	2
	3. XP UI	2	1
	4. Upgrades UI	3	1
	5. Upgrade Integration in Unity	8	
	6. Compatibility Testing	4	1
	7. Error Documentation and Fixin	6	
	8. Artwork	4	
	subtotal	32	8

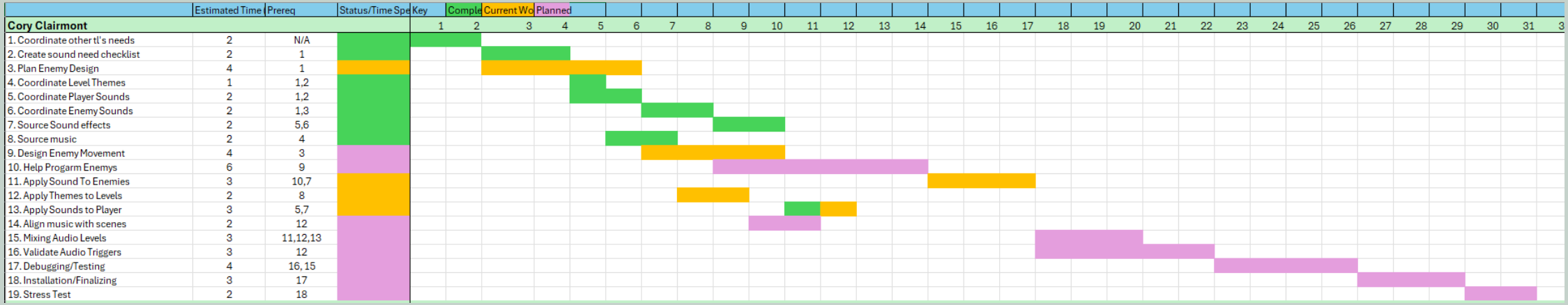
No planned, current, and completed tab. Completed sections don’t follow prerequisites.



Liam	task	Hours to c	spent(hrs)	
	predefine workload	2	1	
	collaboration between members	3	2	
	organization of git files	3	1	1
	character design + theme design	8	1	2,3
	documentation and revision	6		2,4
	programming and haptic integration + mobile	16		1,5
	testing features and capability of mobile devices	4		3,6
	Install and integration	4		5,6,7
	subtotal	48	5	

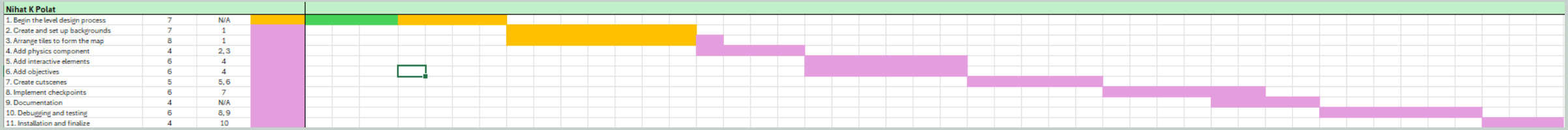
Not too many hours completed, slack doesn't match the prerequisites, 6 must be completed before 8 begins. (don’t let your pipe burst next time, 10 minutes late to presentation unforgivable)

Gantt Roast – Brain Stew



Cory Clairmont	Estimated Time	Prereq
1. Coordinate other tl's needs	2	N/A
2. Create sound need checklist	2	1
3. Plan Enemy Design	4	1
4. Coordinate Level Themes	1	1,2
5. Coordinate Player Sounds	2	1,2
6. Coordinate Enemy Sounds	2	1,3
7. Source Sound effects	2	5,6
8. Source music	2	4
9. Design Enemy Movement	4	3
10. Help Progarm Enemys	6	9
11. Apply Sound To Enemies	3	10,7
12. Apply Themes to Levels	2	8
13. Apply Sounds to Player	3	5,7
14. Align music with scenes	2	12
15. Mixing Audio Levels	3	11,12,13
16. Validate Audio Triggers	3	12
17. Debugging/Testing	4	16, 15
18. Installation/Finalizing	3	17
19. Stress Test	2	18

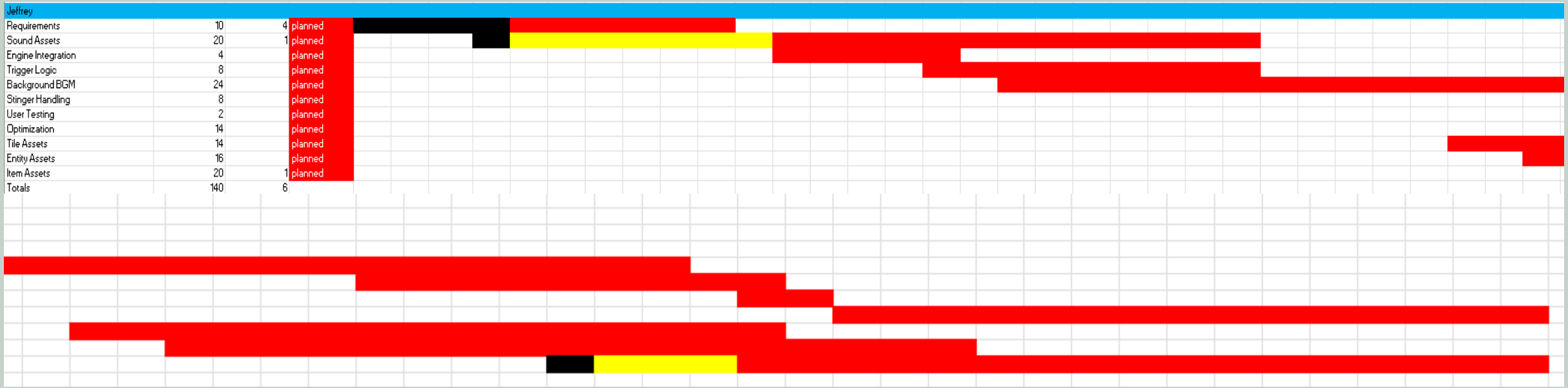
Cory's third task hasn't been completed yet, however it may be due to roadblocks encountered during that allotted dev time period



Nihat K Polat		
1. Begin the level design process	7	N/A
2. Create and set up backgrounds	7	1
3. Arrange tiles to form the map	8	1
4. Add physics component	4	2, 3
5. Add interactive elements	6	4
6. Add objectives	6	4
7. Create cutscenes	5	5, 6
8. Implement checkpoints	6	7
9. Documentation	4	N/A
10. Debugging and testing	6	8, 9
11. Installation and finalize	4	10

Kayra is possibly falling slightly behind, not too much progress made here

Gantt Roast – Spaghetti Studio



Jeffrey			
Requirements	10	4	planned
Sound Assets	20	1	planned
Engine Integration	4		planned
Trigger Logic	8		planned
Background BGM	24		planned
Stinger Handling	8		planned
User Testing	2		planned
Optimization	14		planned
Tile Assets	14		planned
Entity Assets	16		planned
Item Assets	20	1	planned
Totals	140	6	

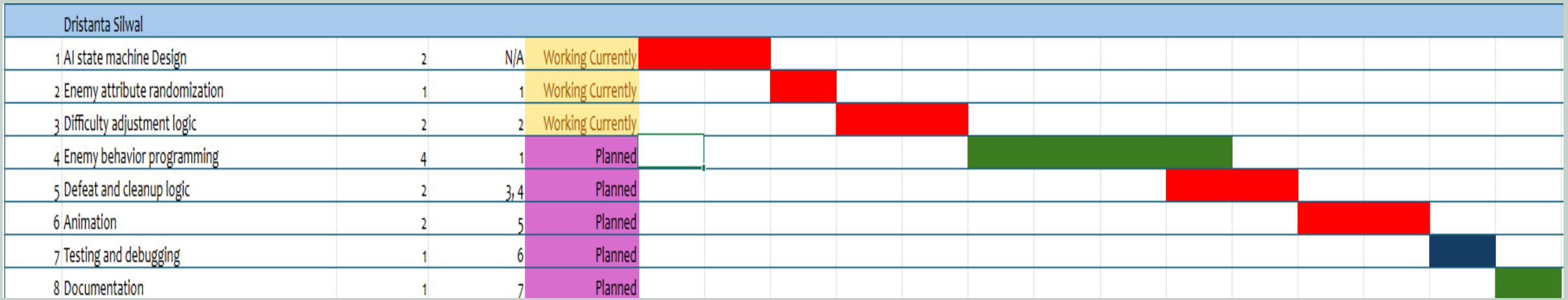
Jeffrey may be falling behind, and the colors hurt my eyes

Garrett			
Weapon Class Creation	4	2	this week
Melee Weapon Creation	15	2	planned
Ranged Weapon Creation	10		planned
Projectile Characteristics	10		planned
Melee Swing Creation	12		planned
Powerup Checks	6		planned
Implementation	3		planned
Testing	6		planned
Totals	66	4	
Group Totals	439	47	
Group Totals ($)	43900	4700	

Garrett's gantt chart shows that he started working on two tasks where one depends on the other's completion

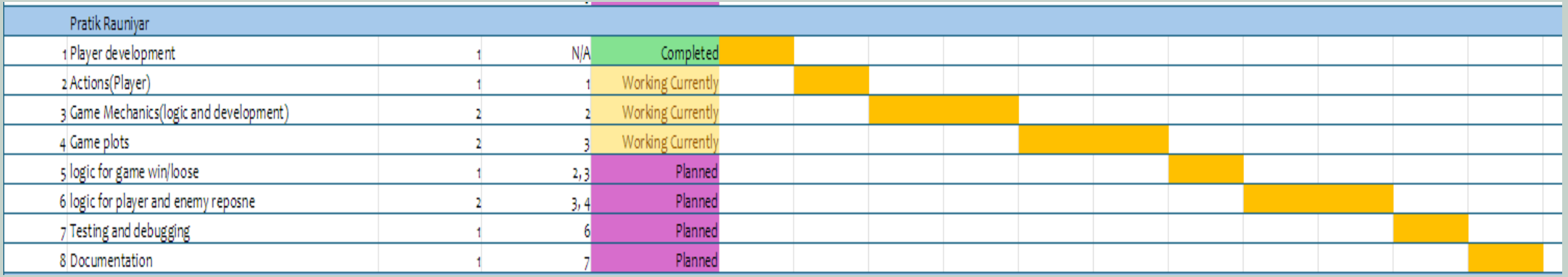
Gantt Roast- Cyber Titans

	Dristanta Silwal			
1	AI state machine Design	2	N/A	Working Currently
2	Enemy attribute randomization	1	1	Working Currently
3	Difficulty adjustment logic	2	2	Working Currently
4	Enemy behavior programming	4	1	Planned
5	Defeat and cleanup logic	2	3, 4	Planned
6	Animation	2	5	Planned
7	Testing and debugging	1	6	Planned
8	Documentation	1	7	Planned



Dristanta has not completed any of his task, might be falling behind.

	Pratik Rauniyar			
1	Player development	1	N/A	Completed
2	Actions(Player)	1	1	Working Currently
3	Game Mechanics(logic and development)	2	2	Working Currently
4	Game plots	2	3	Working Currently
5	logic for game win/loose	1	2, 3	Planned
6	logic for player and enemy reposne	2	3, 4	Planned
7	Testing and debugging	1	6	Planned
8	Documentation	1	7	Planned



Pratik is working on too many features at once but not too much progress.

References

- https://hackernoon.com/grasp-principles-part-3-polymorphism-pure-fabrication-indirection-protected-variations
- https://medium.com/huawei-developers/grasp-principles-that-every-developer-should-know-81d44c684ef9
- https://www.fluentcpp.com/2021/06/23/grasp-9-must-know-design-principles-for-code/
- https://bool.dev/blog/detail/grasp
- https://users.cs.utah.edu/~germain/PPS/Topics/interfaces.html
- https://www.geeksforgeeks.org/grasp-design-principles-in-ooad/
- https://medium.com/wix-engineering/what-exactly-does-low-coupling-high-cohesion-mean-9259e8225372
- https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/
- https://www.engati.com/glossary/cohesion-and-coupling
- https://hackernoon.com/grasp-principles-part-2-controller-low-coupling-and-high-cohesion
- https://www.scaler.com/topics/cohesion-and-coupling-in-software-engineering/