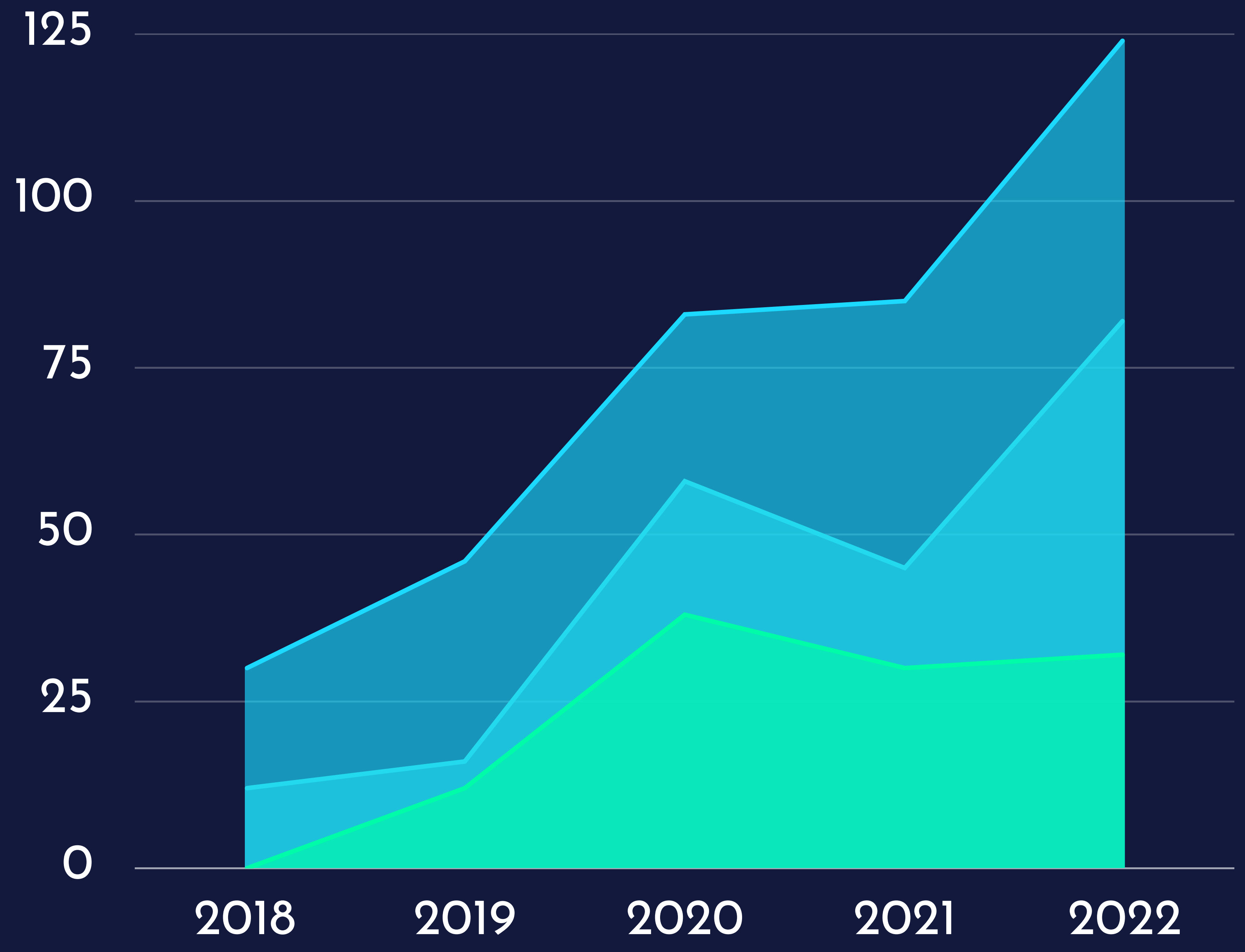
SOLID PRINCIPLES

Ashish Sam T George

Single Responsibility

“THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE”

A class should have only one reason to change. In other words, a class should have only one responsibility or job. This helps in creating modular and focused classes.



Without SOLID

```
public class StorageService {

    public void googleDriveUpload(String data){
        System.out.println("Uploaded to Google 
Drive: " + data);
    }

    public void oneDriveUpload(String data){
        System.out.println("Uploaded to One 
Drive: " + data);
    }
}
```

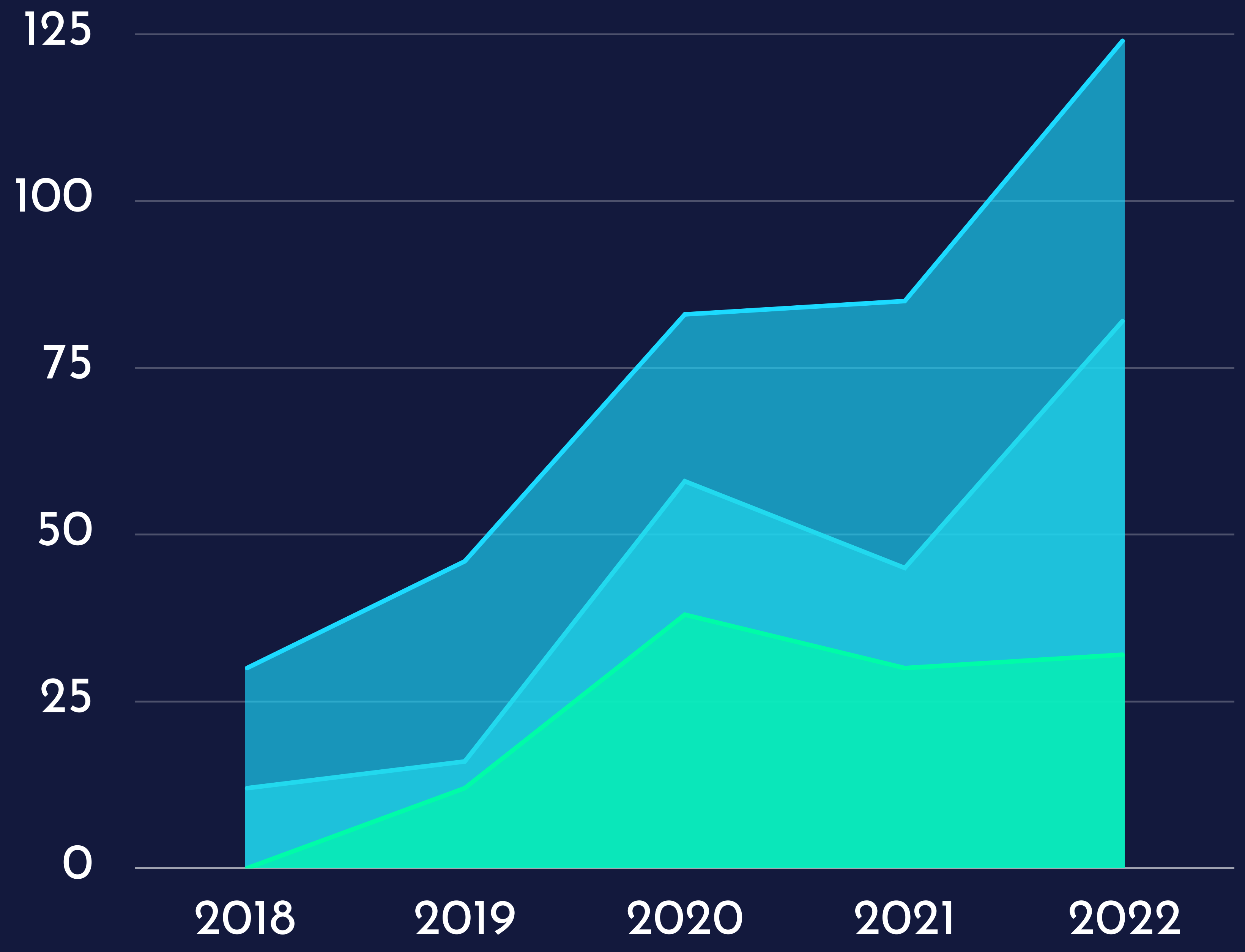
With SOLID

```
public class GoogleDriveStorageService 
implements DataStorable{
    public void upload(String data) {
        System.out.println("Uploaded to Google 
Drive: " + data);
    }
}
public class OneDriveStorageService 
implements DataStorable{
    public void upload(String data) {
        System.out.println("Uploaded to One 
Drive: " + data);
    }
}
```

Open Closed

"SOFTWARE ENTITIES SHOULD BE OPEN FOR EXTENSION, BUT CLOSED FOR MODIFICATION"

Encourages the use of interfaces and abstract classes to allow for future extensions without modifying existing code.



Without SOLID

```
public class FeedbackService {

    public void createImageFeedback(String multimedia){
        System.out.println("Stored the Image: " + multimedia);
    }

    public void createAudioFeedback(String multimedia){
        System.out.println("Stored the Video: " + multimedia);
    }
}
```

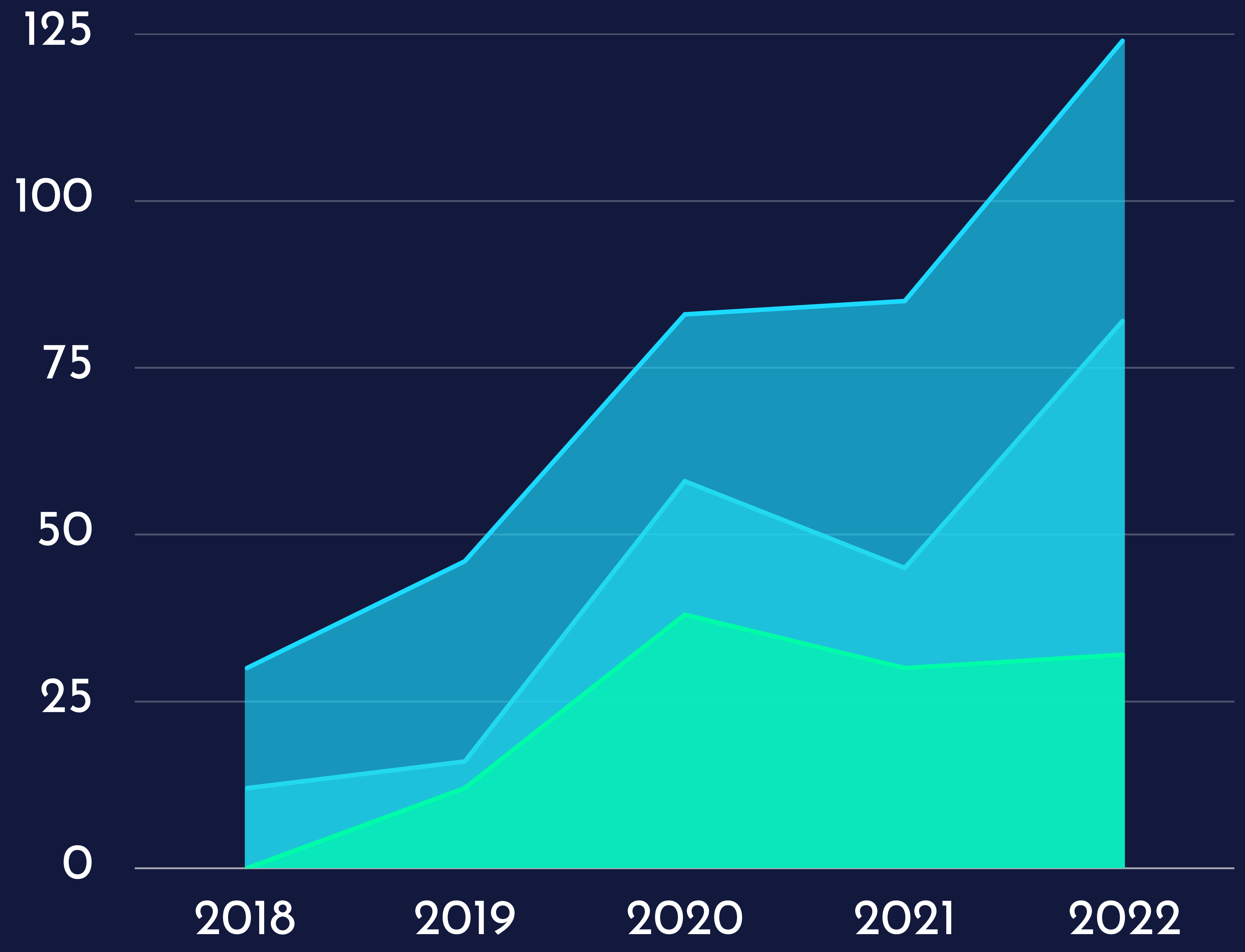
With SOLID

```
public interface FeedbackMultimediaCreatable{
    public String createMultimediaFeedback(String multimedia);
}

public class FeedbackImageService implements FeedbackMultimediaCreatable {
    public String createMultimediaFeedback(String multimedia) {
        String imageUrl = this.feedbackUploadable.upload(multimedia);
        System.out.println("Stored the Image: " + multimedia);
        return imageUrl;
    }
}
public class FeedbackAudioService implements FeedbackMultimediaCreatable {
    public String createMultimediaFeedback(String multimedia) {
        String audioUrl = this.feedbackUploadable.upload(multimedia);
        System.out.println("Stored the Audio: " + multimedia);
        return audioUrl;
    }
}
```

Liskov Substitution

“FUNCTIONS THAT USE POINTERS OR REFERENCES TO BASE CLASSES MUST BE ABLE TO USE OBJECTS OF DERIVED CLASSES WITHOUT KNOWING IT”

Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program. It ensures that derived classes can be true substitutes for their base classes.



Without SOLID

```
public class FeedbackService{

    public void createNormalFeedback(){
        System.out.println("Created Normal Feedback");
    }

    public void createRatingFeedback(){
        System.out.println("Created Rating Feedback");
    }
}
```

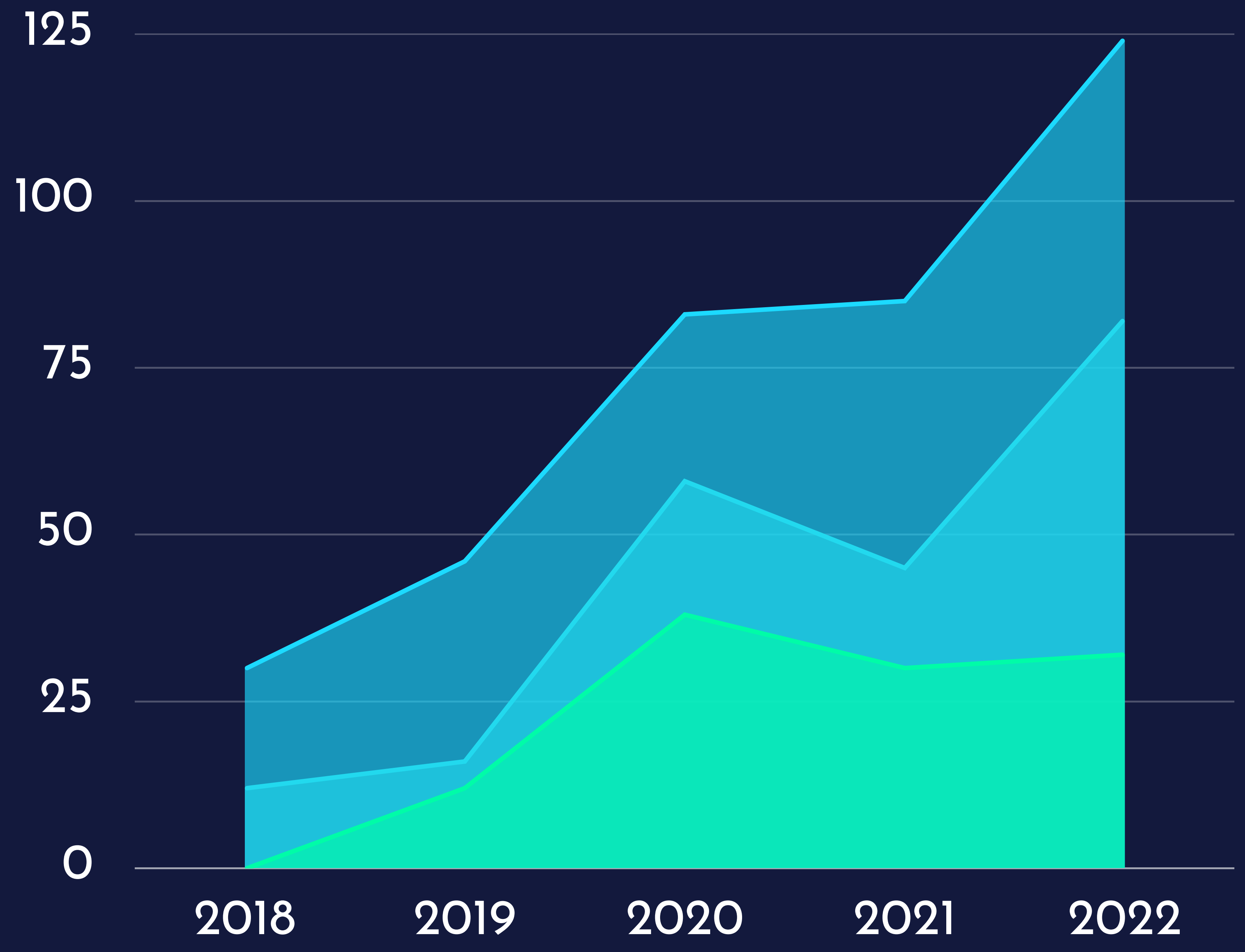
With SOLID

```
public abstract class FeedbackService {
    public abstract void createFeedback(Feedback feedback);
}

public void createFeedback(Feedback feedback) {
    NormalFeedback normalFeedback = (NormalFeedback) feedback;
    System.out.println("Created the feedback: " +
normalFeedback.getFeedbackMessage());
}

public void createFeedback(Feedback feedback) {
    RatingFeedback ratingFeedback = (RatingFeedback) feedback;
    System.out.println("Created the rating: " + ratingFeedback.getRating());
}
```

Interface Seggregation

“CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES THAT THEY DO NOT USE”

It's better to have small, specific interfaces than a large, all-encompassing one.



Without SOLID

```
public interface FeedbackCreatable{
    public void create(Feedback feedback);
    public void multimediaCreate(String multimedia);
}

public class FeedbackService implements FeedbackCreatable {
    public void create(Feedback feedback) {
        System.out.println("Created New Feedback");
    }
    public void multimediaCreate(String multimedia) {
        return;
    }
}
```

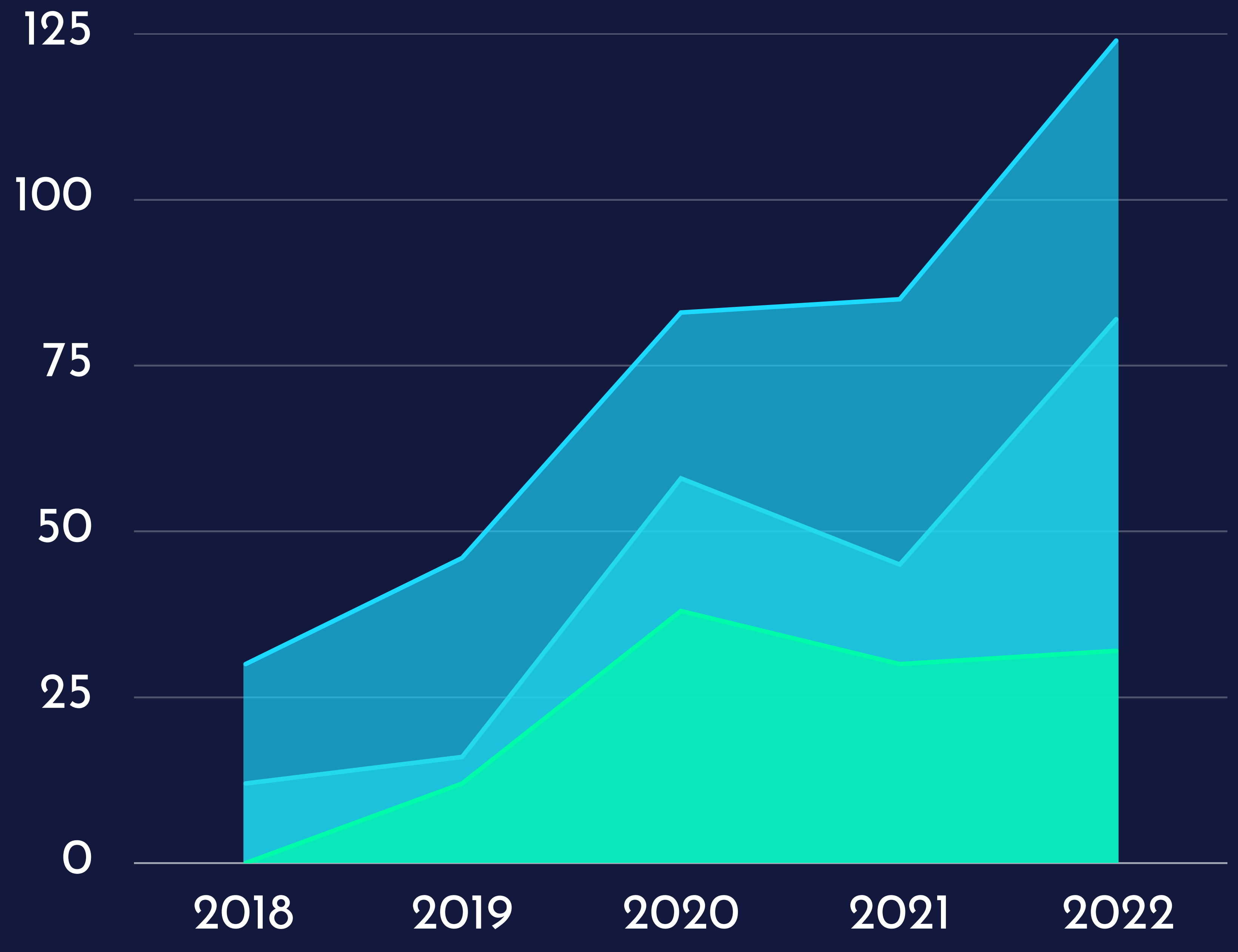
With SOLID

```
public interface FeedbackCreatable {
    public String create(Feedback feedback);
}
public interface FeedbackMultimediaCreatable {
    public String createMultimediaFeedback(String multimedia);
}

public class FeedbackService implements FeedbackCreatable {
    public void create(Feedback feedback) {
        System.out.println("Created New Feedback");
    }
}
public class FeedbackMultimediaService implements FeedbackMultimediaCreatable {
    public void createMultimediaFeedback(String multimedia) {
        System.out.println("Created New Feedback");
    }
}
```

Dependency Inversion

“DEPEND UPON ABSTRACTIONS, [NOT] CONCRETES”

High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.



Without SOLID

```
public class MongoDbRepository{
    public void store(String data) {
        System.out.println("Stored in MongoDB: " + data);
    }
}

public class MySqlRepository{
    public void store(String data) {
        System.out.println("Stored in MySQL Database: " + data);
    }
}

public class FeedbackUtility{
    public void saveFeedback(){
        MongoDbRepository mongoDbRepository = new MongoDbRepository();
        mongoDbRepository.store("Data");
    }
}
```

With SOLID

```
public interface Repository {
    public void store(String data);
}

public class MongoDbRepository implements Repository{
    public void store(String data) {
        System.out.println("Stored in MongoDB: " + data);
    }
}

public class MySqlRepository implements Repository{
    public void store(String data) {
        System.out.println("Stored in MySQL Database: " + data);
    }
}
```