**Synthetic Software Defect Classification Dataset**

These features collectively simulate a diverse set of factors that can contribute to the occurrence of software defects in a real-world scenario.

1. **Complexity**

* Description: Represents the complexity of the software.
* Contribution to Software Defects: Higher complexity may lead to coding errors and increase the likelihood of defects.

1. **Incomplete\_Requirements**

* Description: Indicates whether the project has incomplete requirements.
* Contribution to Software Defects: Incomplete requirements can result in misunderstandings and potentially lead to defects during development.

1. **Poorly\_Defined\_Specifications**

* Description: Indicates whether project specifications are poorly defined.
* Contribution to Software Defects: Poorly defined specifications can lead to incorrect implementations and, consequently, defects.

1. **Coding\_Errors**

* Description: Represents the presence of coding errors.
* Contribution to Software Defects: The existence of coding errors directly contributes to software defects.

1. **Lack\_of\_Code\_Reviews**

* Description: Indicates whether there is a lack of code reviews.
* Contribution to Software Defects: Lack of code reviews may result in undetected defects, as issues could go unnoticed.

1. **Inadequate\_Testing**

* Description: Represents whether testing is inadequate.
* Contribution to Software Defects: Inadequate testing increases the chances of defects going undetected before deployment.

1. **Time\_Pressure**

* Description: Reflects whether there is time pressure on the development process.
* Contribution to Software Defects: Time pressure may lead to rushed development, increasing the likelihood of coding errors and defects.

1. **Inadequate\_Documentation**

* Description: Indicates whether documentation is inadequate.
* Contribution to Software Defects: Inadequate documentation can make it challenging to understand and modify the code correctly, potentially leading to defects.

1. **Communication\_Issues**

* Description: Reflects whether there are communication issues among team members.
* Contribution to Software Defects: Poor communication can result in misunderstandings, leading to defects in the implemented features.

1. **Changing\_Requirements**

* Description: Indicates whether there are frequent changes to project requirements.
* Contribution to Software Defects: Frequent changes can introduce errors if not managed effectively, leading to defects.

1. **Project\_Management\_Metrics**

* Description: Represents project management metrics.
* Contribution to Software Defects: Project management metrics could indirectly influence defects based on factors like adherence to timelines and resource allocation.

1. **Code\_Complexity**

* Description: Reflects the overall complexity of the code.
* Contribution to Software Defects: Higher code complexity can lead to difficulties in understanding and maintaining the code, potentially resulting in defects.

1. **Code\_Review\_Findings**

* Description: Represents the number of findings identified during code reviews.
* Contribution to Software Defects: The number of findings in code reviews is directly correlated with potential defects in the code.

1. **Testing\_Defects**

* Description: Represents the number of defects identified during testing.
* Contribution to Software Defects: Testing defects directly contribute to the count of identified software defects.

1. **Driver\_Fatigue**

* Description: Reflects whether driver fatigue is considered as a factor.
* Contribution to Software Defects: If relevant, driver fatigue could impact the likelihood of defects in software related to driving scenarios.

1. **Vehicle\_Defects**

* Description: Indicates whether there are defects in the vehicle.
* Contribution to Software Defects: Vehicle defects, if applicable to the software, could influence the likelihood of software defects.

1. Weather\_Conditions

* Description: Represents adverse weather conditions.
* Contribution to Software Defects: Adverse weather conditions may impact software related to driving or other weather-sensitive applications.

1. **Defect**

* Description: Binary variable (0: No Defect, 1: Defect).
* Contribution to Software Defects: Represents the presence or absence of software defects, serving as the target variable for classification.

## Assumptions

**Independence of Features**

* Assumption: The features are assumed to be relatively independent, and correlations are introduced explicitly where deemed necessary.

**Linear Relationships**

* Assumption: The relationships between features and the target variable are assumed to be approximately linear.

**Normal Distribution**

* Assumption: The synthetic data assumes a normal distribution for certain features, and deviations from this distribution may occur based on the random generation process.

**Class Imbalance**

* Assumption: The dataset assumes a class imbalance with 80% of samples having no defects (class 0) and 20% having defects (class 1).

**Randomness**

* Assumption: The synthetic data incorporates a level of randomness to simulate real-world variability in software development scenarios.

## **Practical research questions**

The questions aim to explore various aspects of software development and the factors associated with the occurrence of defects. Depending on the specific goals and context, researchers can delve deeper into each question, analyze patterns, and draw meaningful insights from the synthetic dataset.

1. Impact of Code Complexity on Defects
   * How does the complexity of the software code influence the likelihood of defects, and can we identify a threshold complexity level associated with increased defect rates?
2. Effect of Inadequate Testing on Defects
   * To what extent does inadequate testing contribute to the occurrence of defects, and what specific testing practices or strategies are associated with lower defect rates?
3. Role of Communication Issues in Software Defects:
   * How do communication issues among team members correlate with the occurrence of software defects, and what communication strategies can mitigate this impact?
4. Weather Conditions and Driving-Related Defects
   * Are adverse weather conditions significantly linked to defects in software related to driving scenarios, and can specific adaptations in the software reduce this impact?
5. Analysis of Changing Requirements and Defects
   * What is the relationship between frequent changes in project requirements and the occurrence of defects, and how can project management adapt to minimize this impact?
6. Evaluation of Project Management Metrics
   * Do specific project management metrics, such as adherence to timelines and resource allocation, show a statistically significant association with software defects?
7. Assessment of Driver Fatigue Impact on Defects
   * To what extent does driver fatigue, if considered as a factor, influence defects in software related to driving, and can software adaptations mitigate this influence?
8. Effect of Lack of Code Reviews on Defect Rates
   * How does the absence of code reviews correlate with the occurrence of defects, and what practices can be implemented to address this correlation?
9. Relationship Between Coding Errors and Defects
   * What is the strength of the relationship between the presence of coding errors and the occurrence of defects, and can specific coding practices reduce the likelihood of errors?
10. Impact of Time Pressure on Software Defects
    * How does time pressure during the development process contribute to defects, and what time management strategies can be employed to mitigate this impact?

### **Software development and machine learning**

These research questions aim to bridge the gap between traditional software development practices and the application of machine learning techniques. They explore ways to leverage machine learning models to improve software defect prediction, enhance interpretability, and evaluate the effectiveness of ensemble methods in the context of the synthetic dataset. Researchers can experiment with various machine learning algorithms, feature engineering strategies, and model evaluation techniques to address these questions.

1. Optimizing Machine Learning Models for Software Defect Prediction
   * How can machine learning models be optimized to predict software defects based on a combination of traditional software development metrics and features introduced in the dataset, and what is the impact of feature selection or engineering on model performance?
2. Interpretable Machine Learning for Defect Identification
   * Can interpretable/explainable machine learning models provide actionable insights into the specific software development factors that contribute most to the prediction of defects, and how do these insights compare with traditional software engineering wisdom?
3. Ensemble Learning Approaches for Robust Defect Prediction
   * How do ensemble learning approaches, combining multiple machine learning models, enhance the robustness and accuracy of software defect prediction, and can ensemble methods effectively handle the diverse set of features introduced in the synthetic dataset?