Chapter 4
Empirical investigation

- Software engineering investigation
- Investigation principles
- Investigation techniques
- Formal experiments: Planning
- Formal experiments: Principles
- Formal experiments: Types
- Formal experiments: Selection
- Guidelines for empirical research

Software Engineering Investigation

Introduction

- Software Engineering Investigation refers to the **systematic and scientific study of software development processes, tools, and products** to improve the practice of software engineering. It involves identifying problems, collecting data, analyzing results, and forming evidence-based conclusions about software engineering phenomena.
- In simple terms, it is the **research and analysis activity** carried out to understand how software is built, how it behaves, and how its development can be improved.

Definition

- Software engineering investigation can be defined as:

“A structured process of exploring, observing, and evaluating software systems, methods, tools, and practices to generate knowledge and improve software development outcomes.”

It forms the foundation of **empirical software engineering**, where theories and methods are tested through real-world data and systematic observation.

Purpose of Software Engineering Investigation

The main purposes are:

- To **identify and analyze problems** in software development and maintenance.
- To **evaluate tools, techniques, and processes** for effectiveness and efficiency.
- To **collect and interpret data** related to software quality, productivity, and performance.
- To **build evidence-based knowledge** to guide practitioners and researchers.
- To **support decision-making** in selecting software engineering methods and tools.

Types of Software Engineering Investigation

Type	Description	Example
Analytical Investigation	Theoretical or model-based study using mathematical or logical analysis.	Evaluating software reliability models mathematically.
Empirical Investigation	Evidence-based study using experiments, surveys, case studies, or observations.	Measuring defect rates across projects using Agile vs. Waterfall.
Exploratory Investigation	Investigating new or emerging technologies or phenomena.	Exploring the use of AI tools in software testing.
Comparative Investigation	Comparing different approaches or tools.	Comparing performance of different programming paradigms.
Descriptive Investigation	Observing and describing real-world practices without manipulation.	Documenting software maintenance activities in a company.

Steps in Conducting a Software Engineering Investigation

- **Problem Identification**
  - Define the software engineering issue or area to be investigated.
  - Example: “Why does project X have frequent requirement changes?”
- **Literature Review**
  - Study existing research and theories related to the topic.
  - Helps identify gaps and formulate hypotheses.
- **Research Design and Planning**
  - Decide the type of investigation: experiment, case study, survey, etc.
  - Determine the data to be collected (quantitative or qualitative).
- **Data Collection**
  - Gather data from projects, repositories, tools, or developers.
  - Methods: observation, interviews, experiments, or archival analysis.
- **Data Analysis**
  - Use statistical or qualitative methods to interpret data.
  - Look for patterns, correlations, and causal relationships.
- **Validation**
  - Evaluate the credibility and reliability of findings.
  - Ensure results are valid and applicable in similar contexts.
- **Reporting and Conclusion**
  - Present findings clearly, often with recommendations for improvement.

Areas of Investigation in Software Engineering

- Software engineering investigations may target various domains, including:
- **Software Process and Methodologies** (e.g., Agile, DevOps, Waterfall)
- **Software Quality and Testing**
- **Project Management and Estimation**
- **Requirements Engineering**
- **Human Factors in Software Development**
- **Software Metrics and Measurement**
- **Software Maintenance and Evolution**
- **Software Tools and Automation**

Importance of Software Engineering Investigation

- Promotes **scientific understanding** of software practices.
- Improves **software quality, productivity, and predictability**.
- Provides **evidence for process improvement** and decision-making.
- Encourages **innovation and validation** of new software technologies.
- Bridges the gap between **academic research and industrial application**.

Investigation principles

- Investigation principles serve as the **foundation for conducting systematic, valid, and reliable research** in software engineering. They ensure that the study is conducted scientifically, ethically, and transparently so that its findings are trustworthy and can be reproduced or applied in practice.
- In software engineering, these principles guide researchers and practitioners when exploring processes, methods, tools, or human factors in software development.

Meaning of Investigation Principles

- The principles of investigation are **guidelines or fundamental rules** that ensure the investigation:
- is **objective** and **evidence-based**,
- follows a **scientific process**,
- produces **valid and reproducible results**, and
- **respects ethical standards** and professional integrity.

Principles of Software Engineering Investigation

Principle	Description	Outcome
Objectivity	Free from bias	Credible results
Systematic Approach	Step-by-step scientific method	Organized and logical study
Empirical Evidence	Based on real-world data	Evidence-based findings
Reproducibility	Can be repeated by others	Reliability and trust
Validity	Measures what it claims	Accurate conclusions
Reliability	Consistent results	Stable findings
Ethical Conduct	Honest and responsible behavior	Research integrity
Relevance	Practical applicability	Useful outcomes
Measurability	Use of quantifiable metrics	Precision and clarity
Critical Analysis	Logical interpretation	Insightful understanding
Documentation	Clear, complete records	Transparency

Investigation techniques

- An **investigation technique** is a structured method used to **gather information and evidence** about software systems, processes, or human factors.

It determines *how* data is collected (e.g., observation, survey, experiment) and *how* it is analyzed to derive conclusions.

Major Investigation Techniques

- Investigation techniques in software engineering can be broadly classified into **empirical** and **analytical** categories.

A. Empirical Investigation Techniques

- Empirical techniques rely on **observations, measurements, and experimentation** to study software phenomena.
  These are the most common in evidence-based software engineering.

1. Experimentation

- Involves **controlled testing of hypotheses** under defined conditions.
- Researchers manipulate one or more variables to observe their effect.
- Often used to compare tools, techniques, or methods.
  Example: Comparing two debugging tools to see which one reduces defect-fix time.

Key Features:

- Control and treatment groups
- Randomization
- Measurable outcomes

2. Case Study

- A **detailed and contextual investigation** of one or a few software projects or organizations.
- Helps understand **complex real-world situations**.
  Example: Studying how an organization implements Agile methods in a large-scale project.

Advantages:

- Deep understanding of context
- Suitable for exploratory research

3. Survey Research

- Data is collected through **questionnaires or interviews** from individuals or groups (e.g., developers, testers, managers).
- Suitable for studying **opinions, perceptions, or experiences**.
  Example: Surveying software engineers about the challenges of remote teamwork.

Advantages:

- Reaches a large population
- Cost-effective and flexible

4. Observation

- Researchers **observe participants or processes** in their natural environment without interference.
- Used to study actual behavior rather than reported behavior.
  Example: Observing how developers use version control tools in daily work.

Types:

- **Participant observation:** Researcher takes part in the activity.
- **Non-participant observation:** Researcher only watches.

5. Interviews

- Conducting **structured or unstructured conversations** with key participants to collect detailed qualitative data.
  Example: Interviewing project managers about risk management strategies.

Advantages:

- In-depth and flexible
- Useful for understanding motivations and opinions

6. Mining Software Repositories (MSR)

- Involves analyzing **existing software artifacts and repositories** (e.g., GitHub, Jira, Bugzilla).
- Helps identify **patterns, trends, and correlations**.
  Example: Mining commit logs to analyze defect introduction rates.

Advantages:

- Uses real-world data
- Enables large-scale analysis

7. Simulation

- Creating a **model of a software process or system** and experimenting with it virtually.
- Useful when real experimentation is costly or risky.
  Example: Simulating the effect of requirement changes on project delivery time.

B. Analytical Investigation Techniques

- Analytical techniques are based on **theoretical analysis, logical reasoning, and modeling** rather than direct observation.

1. Modeling and Formal Analysis

- Developing **mathematical or logical models** to describe software systems or processes.
  Example: Modeling software reliability using statistical models.

2. Algorithmic Analysis

- Investigating algorithms for **complexity, performance, and correctness**.
  Example: Analyzing the time complexity of sorting algorithms.

3. Theoretical Simulation

- Building analytical models to **simulate performance or behavior** without actual implementation.
  Example: Predicting network throughput based on algorithmic models.

Factors in Choosing Investigation Techniques

When selecting a technique, consider:

- **Nature of the problem** (exploratory, descriptive, or causal)
- **Availability of data** (quantitative vs. qualitative)
- **Resources and time** available
- **Level of control** possible over variables
- **Ethical constraints** and data privacy

Importance of Investigation Techniques

- Provide **structured methods** for software research.
- Enable **data-driven decision-making** in process and product improvement.
- Help in **validating theories and tools** scientifically.
- Facilitate **continuous improvement** in software engineering practice.

Formal experiments: Planning

Definition:
Planning is the first and most important stage of a formal experiment — it defines what, why, and how the experiment will be conducted.

Purpose:
To ensure the experiment is **scientific, unbiased, valid, and repeatable**.

Example:
Testing whether Test-Driven Development (TDD) reduces defects compared to traditional coding.

Importance:

- Prevents bias and errors
- Ensures valid and reliable results
- Supports evidence-based conclusions

Main Steps in Planning:

- **Define Objectives & Hypotheses** – What question are you answering?
- **Identify Variables** – Independent (cause), Dependent (effect), Controlled.
- **Select Subjects** – Who will participate; ensure fairness and randomization.
- **Design the Experiment** – Choose between-subjects, within-subjects, or factorial design.
- **Prepare Materials & Tools** – Tasks, datasets, software, or measurement instruments.
- **Plan Data Collection** – What data to collect and how.
- **Plan Data Analysis** – Select appropriate statistical or qualitative methods.
- **Address Validity & Reliability** – Ensure results are accurate and reproducible.
- **Ethical Considerations** – Informed consent, confidentiality, honesty.
- **Schedule & Resources** – Time, cost, and roles defined.

Formal experiments: Principles

Formal experiments in software engineering are **controlled investigations** designed to test hypotheses about tools, techniques, or processes.

The **principles** of formal experiments ensure that the study is **scientific, valid, unbiased, and reproducible**.

Importance

- Ensures **scientific accuracy and credibility**.
- Produces **reliable, reproducible results**.
- Supports **evidence-based improvements** in software engineering.
- Encourages **ethical and professional research practices**.

Example

- When testing two debugging tools:
- Randomly assign developers to each tool (randomization).
- Keep project type constant (control).
- Measure defect-fix time using the same method (precision).
- Allow repetition by others (replication).

Key Principles of Formal Experiments

- **Objectivity**
  - The experiment must be free from personal bias.
  - Results should depend on evidence, not opinions.
- **Control**
  - Keep all variables constant except the one being studied (independent variable).
  - Ensures that observed changes are due to the experimental treatment only.
- **Randomization**
  - Subjects or samples are assigned randomly to groups.
  - Helps eliminate selection bias.
- **Replication**
  - The experiment should be repeatable by others with similar results.
  - Increases reliability and confidence in findings.
- **Precision in Measurement**
  - Clearly define how variables are measured.
  - Use accurate and consistent measurement tools.

- **Validity**
  - Ensure results truly reflect the relationship being studied.
  - Includes internal, external, construct, and conclusion validity.
- **Ethical Conduct**
  - Respect participants' rights and confidentiality.
  - Report results honestly and transparently.
- **Statistical Rigor**
  - Use proper statistical analysis to draw valid conclusions.
  - Avoid misinterpretation of data.
- **Documentation and Transparency**
  - Record procedures, data, and analysis clearly.
  - Allows others to evaluate and replicate the study.
- **Feasibility and Relevance**
- The experiment should be practical and address meaningful software engineering problems.

Formal experiments: Types

Formal experiments in software engineering are conducted to **test hypotheses** under controlled conditions.
They differ based on **how subjects and variables are arranged** during the experiment.

Importance

- Helps choose the **right level of control vs. realism**.
- Ensures **scientific accuracy** while maintaining **practical relevance**.
- Allows replication and validation of results under different conditions.

Main Types of Formal Experiments

1. Controlled Experiment

- The most common type.
- Researcher **manipulates one or more independent variables** and observes their effect on dependent variables.
- Includes **control group** (no treatment) and **experimental group** (with treatment).

Example: Comparing code quality between teams using Agile vs. Waterfall methods.

2. Quasi-Experiment

- Similar to controlled experiment but **lacks full randomization**.
- Used when random assignment of subjects is not possible (e.g., in real industrial settings).

Example: Comparing productivity of two existing project teams in a company.

3. Field Experiment

- Conducted in a **real-world environment** rather than a lab.
- Balances realism and control.

Example: Testing a new software tool with developers in an actual software company.

4. Laboratory Experiment

- Conducted in a **controlled, artificial environment** (e.g., university lab).
- Allows **maximum control** over variables.

Example: Students performing tasks with different debugging tools in a lab.

5. Natural Experiment

- The researcher **does not manipulate variables**; they observe natural differences that occur.
- Used when direct control is not possible.
- **Example:** Studying performance differences between teams that naturally use different IDEs.

Formal experiments: Selection

Selection is the process of **choosing the subjects, variables, tools, and design** for a formal experiment.
It ensures that the experiment is **fair, unbiased, and scientifically valid**.

Importance

- Reduces **bias** and **confounding factors**.
- Ensures **valid and reliable** results.
- Makes the experiment **repeatable** and **credible**.

Example

- To test two debugging tools:
- Randomly select developers.
- Assign one group to Tool A and another to Tool B.
- Keep task, time, and environment the same for both groups.

Key Elements of Selection

Selection of Subjects

- Choose participants who represent the target group (e.g., students, developers).
- Use **random selection or random assignment** to reduce bias.
- Ensure equal skill level and experience among groups.

Selection of Variables

- **Independent Variable:** The factor being changed (e.g., testing method).
- **Dependent Variable:** The result measured (e.g., number of bugs).
- **Controlled Variables:** Factors kept constant (e.g., same tools, same task).

Selection of Experimental Design

- Choose how the experiment will be organized:
  - *Between-Subjects* – different groups get different treatments.
  - *Within-Subjects* – same subjects get all treatments.
  - *Factorial* – multiple variables tested together.

Selection of Tools and Environment

- Use the same tools, systems, and settings for all subjects to ensure fairness.

Selection Criteria

- Define who can participate and who cannot.
- Keep the experiment consistent and realistic.

Guidelines for empirical research

- Empirical research in software engineering involves **systematic observation, measurement, and experimentation** to collect real evidence and validate hypotheses.

Importance

- Ensures **scientific quality and credibility**.
- Produces **reliable, valid, and generalizable** results.
- Supports **evidence-based improvement** in software engineering practices.

Example

- If researching the effect of agile methods on defect rates:

  Define hypothesis → Plan experiment → Collect real project data →

  Analyze → Report results honestly.

Main Guidelines

- **Define Clear Objectives**
  - Clearly state the research question or hypothesis.
  - Example: *Does pair programming improve code quality?*
- **Conduct a Thorough Literature Review**
  - Study previous research to understand what is already known.
  - Identify gaps your study will address.
- **Choose an Appropriate Research Method**
  - Select from **experiments, case studies, surveys, or observations** based on your goal.
  - Ensure the method fits the question.
- **Design the Study Carefully**
  - Plan variables, data collection, and analysis before starting.
  - Avoid bias and ensure repeatability.
- **Select Participants Properly**
  - Choose subjects that represent the real-world population (e.g., students, developers).
  - Use randomization where possible.

- **Ensure Validity and Reliability**
  - **Validity:** Are you measuring what you intend to measure?
  - **Reliability:** Will the results be consistent if repeated?
- **Collect Data Systematically**
  - Use reliable tools and consistent procedures.
  - Record all conditions and observations accurately.
- **Analyze Data Objectively**
  - Apply proper **statistical or qualitative analysis**.
  - Avoid manipulating data to fit expectations.
- **Interpret Results Carefully**
  - Link results to research questions.
  - Acknowledge limitations and possible sources of error.
- **Report Transparently**
  - Describe all steps, methods, and data clearly.
  - Allow others to replicate and verify the findings.
- **Follow Ethical Standards**
  - Obtain consent, protect privacy, and report truthfully.
  - Avoid plagiarism or data falsification.