Milestone 7: Validation and Verification Guide

KSU CCSE
CS 4632: Modeling and simulation

Objective

Milestone 7: Validation and Verification focuses on ensuring that your simulation model is both accurate and correctly implemented. Building upon the work completed in Milestone 6, this milestone is a critical step toward finalizing your simulation for submission.

The Validation and Verification (V&V) process involves two main tasks:

- **Validation**: Ensuring the model represents the real-world system accurately by comparing simulation outputs with real-world data, expert expectations, or similar models. This step includes various validation techniques such as parameter validation, historical data validation, and cross-model validation.
- **Verification**: Ensuring the model is implemented correctly and operates as intended. This involves unit testing, integration testing, regression testing, and edge case testing to identify and fix any logical or coding errors in the model.

Students are expected to provide a professional report that documents the V&V process. The report should include:

- A brief introduction that links back to Milestone 6 and explains the importance of V&V.
- Detailed validation methods and results, including documentation of any discrepancies and how they were addressed.
- Thorough verification methods, including testing strategies, results, and resolutions of any identified issues.
- A checklist showing the completion of all validation and verification steps.
- Reflection on common pitfalls encountered and how they were avoided or addressed.
- Proper documentation of assumptions, test cases, and model limitations.
- A conclusion summarizing the readiness of the model for the final deliverable.

This milestone aims to prepare your simulation for its final submission by identifying and resolving potential issues, and ensuring the model is accurate, reliable, and ready for use. By thoroughly completing the V&V process, you will build confidence in your model's correctness and ability to simulate real-world scenarios effectively.

Contents

Introduction

Building upon the work completed in **Milestone 6**, where you conducted sensitivity and scenario analyses, **Milestone 7** focuses on the *validation* and *verification* (V&V) of your simulation model. This step is crucial to ensure that your model accurately represents the real-world system and functions correctly.

Validation answers the question, *"Are we building the right model?"*, while **Verification** answers, *"Are we building the model right?"* This guide provides a systematic approach to performing V&V effectively, ensuring your simulation is reliable, accurate, and ready for the final submission.

As this milestone precedes the final deliverable, allocating sufficient time and effort to V&V is essential. Typically, validation and verification may require substantial but manageable time compared to Milestone 6. Plan accordingly to ensure all aspects are thoroughly addressed and that your model stands up to rigorous scrutiny.

By the end of this milestone, you will have thoroughly validated your model against real-world scenarios and verified its correctness, preparing it comprehensively for presentation and the final report.

1 Validation

1.1 Purpose of Validation

Validation ensures that your simulation model accurately represents the real-world system or process it is intended to simulate. It involves comparing model outputs with real-world data or expert expectations to confirm its accuracy.

1.2 Methods of Validation

1.2.1 Parameter Validation

Check whether the model parameters are within reasonable and acceptable ranges based on real-world data or literature.

- Compare model parameters with empirical values from relevant case studies.
- Use statistical techniques (e.g., confidence intervals, standard deviations) to justify parameter choices.
- Example: If modeling a queuing system, validate arrival rates against known service times.

1.2.2 Historical Data Validation

Compare the simulation outputs with historical data to validate the model's ability to replicate past behavior.

- Gather real-world historical data that aligns with the simulation's scope.
- Run simulations using past input conditions and compare outcomes with recorded historical values.
- Example: In a financial model, compare stock price movement predictions against historical price data.

1.2.3 Cross-Model Validation

Compare your model's results with those from similar published models or simulations in the literature to assess consistency.

- Identify existing models that address similar scenarios and compare output trends.
- Use benchmarking techniques to analyze discrepancies and explain differences.
- Example: A traffic simulation model can be validated by comparing its congestion predictions with results from established transportation models.

1.2.4 Face Validation

Compare the model's behavior with expert opinions or known behavior patterns of the real system.

- Engage subject-matter experts to review the model's logic and behavior.
- Ensure that simulation results align with intuitive or expected real-world patterns.
- Example: A hospital capacity model should reflect expected patient flow dynamics when reviewed by healthcare administrators.

1.2.5 Sensitivity Analysis Follow-up

Use insights from Milestone 6 to see if the model responds to parameter changes as expected.

- Identify parameters that exhibit the highest sensitivity and test their impact.
- Ensure that slight variations in input parameters result in logically consistent changes in outputs.
- Example: If adjusting a resource allocation parameter drastically changes system performance, further investigation may be needed.

1.3 Guidance on Validation

1.3.1 Documenting Validation Results

- Use tables, graphs, and statistical tests to present validation results.
- Clearly label all visuals and include units of measurement.
- Include explanations for each result, highlighting how they support the model's validity.

1.3.2 Addressing Discrepancies

- Identify any differences between simulation outputs and real-world data.
- Analyze the causes of discrepancies (e.g., model assumptions, data inaccuracies).
- Adjust the model as necessary and re-validate.

1.3.3 Acceptable Margins of Error

- Define acceptable error ranges based on industry standards or the specific context of your simulation.
- **Examples:**
  - For financial simulations, an error margin of $\pm 5\%$ may be acceptable.
  - In engineering models, stricter margins such as $\pm 1\%$ might be required.
- Justify the margins of error you consider acceptable.

2 Verification

2.1 Purpose of Verification

Verification ensures that the simulation model is implemented correctly and operates as intended without any coding or logical errors. It focuses on verifying the correctness of individual components, their integration, and the overall stability of the system. By applying systematic testing and review methodologies, verification ensures that the model adheres to its design specifications and functions as expected under different conditions.

2.2 Methods of Verification

2.2.1 Unit Testing

Unit testing validates individual components or functions of your simulation code in isolation. This ensures that each unit performs its intended task correctly before integration with other components.

- Test core computational functions (e.g., random event generators, statistical calculations).
- Use automated testing frameworks such as `unittest` (Python), JUnit (Java), or Google Test (C++).
- Example: Testing a function that calculates an entity's movement within a simulation environment.

2.2.2 Integration Testing

Integration testing evaluates how different modules or components interact within the simulation. The goal is to ensure seamless communication and data exchange across different parts of the system.

- Verify that event queues, resource management, and state transitions work as expected.
- Test interactions between models, input/output modules, and any database or external APIs.
- Example: Testing how a simulated agent interacts with an event scheduler in a discrete-event model.

2.2.3 Regression Testing

Regression testing ensures that recent changes, updates, or bug fixes have not introduced new issues into previously working components.

- Re-run unit and integration tests after modifying code.
- Maintain a set of reference outputs to compare against expected results.
- Example: Verifying that changes to a probability function do not affect previous statistical distributions.

2.2.4 Edge Case Testing

Edge case testing assesses the simulation's behavior under extreme, unexpected, or boundary conditions.

- Test scenarios with extreme values (e.g., zero input, maximum capacity, negative values).
- Evaluate system stability when exposed to unexpected inputs or high computational loads.
- Example: Checking if a queue-based simulation correctly handles an empty queue or an overflow scenario.

2.2.5 Code Review

Peer code review is an essential step in verification, allowing for the detection of logical errors, inefficiencies, and adherence to best practices.

- Perform structured code reviews using guidelines for readability, efficiency, and correctness.
- Use static code analysis tools (e.g., Pylint for Python, Checkstyle for Java, Clang for C++).
- Example: Reviewing a scheduling algorithm to ensure fair and unbiased task allocation.

2.2.6 Version Control Best Practices

Utilizing a structured version control approach ensures traceability, collaboration, and rollback capabilities in case of errors.

- Commit changes frequently with meaningful messages.
- Use branches for feature development and bug fixes.
- Implement continuous integration (CI) pipelines to automate testing upon each commit.
- Example: Using Git with a pull request workflow to review and test code before merging into the main branch.

2.3 Guidance on Verification

2.3.1 Common Verification Pitfalls to Avoid

- Overlooking error messages or warnings during execution.
- Not maintaining sufficient test coverage across components.
- Relying solely on manual testing rather than automation.
- Failing to test under real-world conditions or edge cases.

2.3.2 Documenting Testing Results

- Record test cases, inputs, expected outputs, and actual results.
- Maintain a test log to track issues and resolutions.
- Use tables and structured reports for clarity.

2.3.3 Using Debugging Tools Effectively

- Use built-in debugging tools such as `pdb` (Python), `gdb` (C++), and IDE debuggers.
- Implement logging mechanisms to capture execution flow and identify anomalies.
- Example: Using a logging framework to track unexpected behaviors in event-driven simulations.

2.3.4 Example Code Snippet for Unit Testing

```
import unittest

class TestSimulationFunctions(unittest.TestCase):
    def test_calculate_growth(self):
        result = calculate_growth(100, 0.05)
        self.assertEqual(result, 105)

if __name__ == '__main__':
    unittest.main()
```

Listing 1: Example of a Unit Test Function in Python

3 Validation and Verification Checklist

Use the following checklist to ensure thorough validation and verification:

3.1 Validation Checks

- Have you compared simulation outputs with real-world data?
- Are all model parameters within reasonable and justified ranges?
- Does the model's behavior align with expert expectations?
- Have you documented all validation results using appropriate visuals?
- Have you performed historical data validation?
- Did you conduct cross-model validation where applicable?

3.2 Verification Checks

- Have you tested the model with boundary and edge conditions?
- Did you verify that all input parameters are properly handled?
- Have you checked for mathematical and logical consistency in your code?
- Is the model's behavior reasonable across different time scales or scenarios?
- Have you systematically documented all testing results?
- Did you perform unit, integration, and regression testing?
- Are you following version control best practices?

3.3 Documentation Checks

- Is all documentation up to date and reflective of the current model?
- Have you documented all assumptions and limitations?
- Are test cases and results properly recorded?

3.4 Version Control Checks

- Are all changes committed with descriptive messages?
- Have you tagged versions appropriately?
- Is the repository synchronized with the remote server?

3.5 Peer Review Checks (if applicable)

- Has the code been reviewed by peers or mentors?
- Have you incorporated feedback from reviews?

3.6 Data Validation Completeness Checks

- Is the data used for validation complete and reliable?
- Have you checked for data inconsistencies or anomalies?

4 Common Pitfalls

4.1 Validation Mistakes

- **Ignoring Data Discrepancies**: Failing to address significant differences between model outputs and real-world data.
- **Inadequate Data**: Relying on insufficient or outdated data for validation.
- **Overlooking Parameter Relevance**: Not validating all critical parameters that significantly affect the model.
- **Over-Reliance on a Single Validation Method**: Using only one method may not reveal all issues.
- **Failure to Cross-Validate**: Not comparing model results with similar simulations or empirical data.
- **Misinterpretation of Validation Metrics**: Misusing statistical validation tests or ignoring confidence intervals.

4.2 Verification Oversights

- **Not Documenting Failed Tests**: Ignoring or not recording failed test cases.
- **Insufficient Test Coverage**: Failing to test all parts of the code thoroughly.
- **Assuming Correctness**: Assuming the model is correct without thorough testing due to overconfidence.
- **Lack of Systematic Approach**: Not having a structured plan for fixing identified issues.
- **Skipping Edge Cases**: Not testing how the model performs under extreme conditions.
- **Overlooking Regression Testing**: Not verifying that recent code changes do not break previously validated functionality.

4.3 Avoiding Overfitting in Validation

- Ensure that the model is generalizable and not just tailored to fit the validation data.
- Use separate datasets for validation and testing.
- Perform sensitivity analysis to understand how parameter variations affect model performance.
- Evaluate model robustness by testing against real-world scenarios beyond the training dataset.

4.4 Tips for Efficient Testing Strategies

- Prioritize testing of critical components that have the most impact.
- Automate repetitive tests where possible.
- Keep detailed records to track changes and outcomes.
- Use structured test case templates to ensure comprehensive test coverage.
- Incorporate continuous integration (CI) tools to run automated tests for each update.
- Utilize debugging and profiling tools to identify performance bottlenecks.

5 Documentation Requirements

5.1 Assumptions Documentation

- Clearly state any assumptions made during validation and verification.
- Justify why each assumption is reasonable, referencing relevant literature or empirical data.
- Document any simplifications or constraints imposed on the model.

5.2 Reporting Validation Results

- Use standardized templates for consistency (see Appendix 7.2).
- Include all relevant data, calculations, and statistical analyses.
- Highlight how results support the model's validity and specify acceptable margins of error.
- Use tables and graphs to clearly present validation findings.

5.3 Guidelines for Error Documentation

- Record all errors encountered, including steps to reproduce and resolve them.
- Categorize errors by severity (e.g., minor, critical) and type (e.g., logic, numerical instability, input errors).
- Maintain an error log with timestamps and resolutions to track recurring issues.
- Include screenshots or debugging logs when applicable.

5.4 Documenting Model Limitations

- Identify any limitations discovered during V&V, such as data availability, computational constraints, or untested edge cases.
- Explain the potential impact of these limitations on the model's outcomes and decision-making reliability.
- Suggest possible ways to address or mitigate these limitations in future iterations.
- Indicate whether the limitations affect validation, verification, or both.

5.5 Format for Reporting Test Cases

- Use a consistent format for all test cases (refer to Appendix 7.1).
- Include test case ID, description, input data, expected results, actual results, and status.
- Ensure that each test case references its corresponding validation or verification method.

- Maintain a version-controlled test repository for tracking changes and updates.

6 Deliverable Requirements

6.1 What to Submit

- A **single PDF document** generated from LaTeX.

6.2 File Naming Format

- `CS_4632_[FirstName]_[LastName]_Milestone_7.pdf`

6.3 Document Contents

Your report should include the following sections:

1. **Introduction**
   - Connection to Milestone 6 and the importance of V&V.
   - Timeline perspective and effort expectations.
2. **Validation**
   - Methods used (Parameter Validation, Historical Data Validation, Cross-Model Validation, etc.).
   - Documentation of validation results (tables, statistical tests).
   - Discussion of any discrepancies and how they were addressed.
   - Acceptable margins of error and justification.
3. **Verification**
   - Methods used (Unit Testing, Integration Testing, Regression Testing, Code Review).
   - Documentation of testing results.
   - Discussion of any issues found and resolutions.
   - Version control practices followed.
4. **Validation and Verification Checklist**
   - Completed checklist from Section 3.
5. **Common Pitfalls**
   - Reflection on how you avoided common mistakes.
6. **Documentation**
   - Assumptions made during V&V.
   - Format for reporting test cases.
   - Presentation of comparison results.
   - Documentation of model limitations.
7. **Conclusion**
   - Summary of the V&V process and readiness for the final deliverable.

6.4 Formatting Guidelines

- You may use **single-column or double-column** format. - Include tables and code snippets as needed to effectively present your results. - Ensure all visuals are properly labeled and referenced in the text. - Follow professional report formatting standards.

6.5 Grading Criteria

Your submission will be evaluated based on the following criteria (Total: 50 Points):

Table 1: Grading Criteria for Milestone 7

Criteria	**Points**	**Minimum Requirement**
Validation Methods and Results	15	Adequate use of validation methods and proper do
Verification Methods and Results	15	Thorough verification with documented re
Documentation Quality	10	Clear, organized, and complete documenta
Presentation and Formatting	5	Professional formatting and adherence to gui
Reflection on Common Pitfalls	5	Insightful reflection on avoiding pitfall
Total	**50**	

6.6 Extra Credit Opportunities

- Up to 5 extra points for exemplary work, such as:

- Implementing advanced validation or verification techniques.
- Exceptional documentation and presentation quality.

7 Example Templates and References

7.1 Test Case Documentation Template

Table 2: Test Case Documentation Template

Test Case ID	TC-01
Description	Verify that the model correctly handles zero input values for parameter X.
Input Data	Parameter X = 0
Expected Result	The model should return a default value without error.
Actual Result	[To be filled after testing]
Status	Pass/Fail

7.2 Validation Results Reporting Template

Table 3: Validation Results Template

Parameter	**Real-World Data**	**Simulation Output**	**Error (%)**
Parameter A	100	98	2%
Parameter B	200	210	5%
Parameter C	150	147	2%

7.3 List of Recommended Tools for V&V

- **Version Control**: Git, GitHub, GitLab
- **Unit Testing Frameworks**:
  - Python: `unittest`, `pytest`
  - Java: JUnit
  - C++: Google Test
- **Debugging Tools**:
  - IDE Debuggers (e.g., Visual Studio Code, PyCharm)
  - Logging libraries
- **Data Analysis and Visualization**:
  - Python: Matplotlib, Pandas
  - R: ggplot2

8 References to Validation and Verification Methods

Depending on your simulation type, you may find the following methods appropriate:

- **Discrete Event Simulations**: Use event validity checks and time progression verification.
- **Agent-Based Models**: Validate agent behaviors and interactions; use micro-level validation techniques.
- **System Dynamics Models**: Verify stock and flow consistency; use dimensional consistency checks.
- **Monte Carlo Simulations**: Validate statistical outputs; check for convergence.

Recommended Readings:

- Law, A. M. (2015). *Simulation Modeling and Analysis.* McGraw-Hill Education.
- Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of Simulation*, 7(1), 12-24.

9 Conclusion

Validation and verification are critical steps in ensuring your simulation model is accurate, reliable, and ready for real-world application. By thoroughly validating your model against real-world data and carefully verifying the correctness of its implementation, you strengthen the integrity of your simulation results. This milestone provides the opportunity to identify and address any discrepancies, improving the overall robustness of the model before final submission.

A well-documented validation and verification process ensures transparency and reproducibility, both of which are essential for high-quality simulation modeling. Your work in this milestone directly contributes to the credibility of your final deliverable.
Final Reminders:

- Ensure all validation results are supported with statistical or empirical justification.
- Verify that your model behaves as expected under different test conditions.
- Maintain structured and organized documentation to support reproducibility.
- Reflect on any challenges faced and document your solutions for future reference.
- Prepare your final report in a professional, clear, and structured manner.

By completing this milestone, you lay the groundwork for a successful final submission, demonstrating both technical competence and analytical rigor.

Appendix A: Vocabulary and Definitions

This appendix provides a glossary of key terms related to **Verification** and **Validation** in the context of **Modeling and Simulation**. Familiarity with these concepts is essential for completing Milestone 7.

1. Verification

Verification ensures that the simulation model has been correctly implemented according to its specifications, free from errors or bugs. It asks the question: **"Are we building the model right?"**

Key Concepts:

- **Unit Testing**: Testing individual components of the model to verify that each part functions as expected.
- **Integration Testing**: Testing the interaction between integrated components.
- **Regression Testing**: Ensuring that new changes do not break existing functionalities.
- **Edge Case Testing**: Examining the model's performance under extreme or boundary conditions.
- **Code Review**: Systematic examination of the simulation code to identify potential errors.

2. Validation

Validation ensures that the simulation model accurately represents the real-world system it is designed to simulate. It asks the question: **"Are we building the right model?"**

Key Concepts:

- **Face Validation**: Reviewing the model's behavior to determine if it intuitively appears correct.
- **Historical Data Validation**: Comparing the simulation outputs with historical data.
- **Cross-Model Validation**: Comparing results with similar published models.
- **Parameter Validation**: Checking whether input parameters are realistic and within expected ranges.
- **Sensitivity Analysis Validation**: Ensuring expected behaviors when parameters vary.

3. General Terms

Key Concepts:

- **Simulation Fidelity**: The degree to which a simulation model accurately reproduces the real-world system.
- **Assumptions**: Underlying factors that the model relies on but are not explicitly verified.
- **Boundary Conditions**: Constraints or limits applied to the model.
- **Stochastic Models**: Models that incorporate randomness.
- **Deterministic Models**: Models that produce the same output given the same input.
- **Convergence Testing**: Testing whether repeated simulations produce stable results.
- **Overfitting**: When a model is too closely aligned to specific data, reducing its generalizability.

4. V&V Process Terms

- **Traceability**: Ability to track each requirement through development and testing.
- **Test Case**: A specific scenario used to verify a component of the model.
- **Error Margins**: Acceptable range of deviations between predictions and real data.
- **Model Tuning**: Adjusting parameters to improve alignment with data.
- **Iterative Refinement**: Gradually improving the model through repeated testing and adjustment.

Formatting Notes

- Use the `fancyhdr` package for customizing headers and footers.
- Include the `hyperref` package for cross-referencing.
- Use `\label{}` and `\ref{}` to create cross-references.
- Ensure all sections and subsections are properly numbered and labeled.
- Use the `graphicx` package to include images (replace with tables if images are not available).
- Use the `listings` package for code snippets.
- Follow LaTeX best practices for a clean and professional document.