Presenter Notes for Leiden University Job Talk

Domain-Specific Solutions for API Misuse
Duration: 30 minutes | Date: July 3, 2025

Slide 1: Title Slide (30 seconds)

- **Opening**: "Good morning everyone. Thank you for having me here today."
- **Purpose**: "I'm excited to share my work on making APIs fundamentally safer to use, and my vision for how this approach can guide us toward trustworthy AI-enhanced software engineering."
- **Personal touch**: "I'm particularly thrilled about the potential collaborations here at LIACS."

Slide 2: Agenda (30 seconds)

- **Set expectations**: "We'll spend about 15 minutes on the concrete problem of API misuse and my solution, jGuard"
- **Vision preview**: "Then 10 minutes on how these principles extend to the broader challenges of AI-assisted programming"
- **Collaboration**: "And finally, specific ways my work aligns with LIACS research"
- **Quick pace**: Don't linger - this is just the roadmap

Slide 3: Challenges (45 seconds)

- **Define API misuse**: "Any incorrect usage that compilers can't catch but causes real problems"
- **Point to image**: "This simple initialization example shows how many places things can go wrong"
- **Research backing**: "Studies show even experienced developers struggle with correct API usage"
- **Transition**: "But what's the real impact of these misuses?"

Slide 4: The API Misuse Crisis (1 minute)

- **Impact number**: Let the $85 Million figure glow for effect
- **Zoom example**: "Not just a fine - real security vulnerabilities affected millions"
- **Walk through flow**: Point to each node in diagram
- **Why it matters**:
  - "APIs are everywhere - cloud, mobile, IoT"
  - "Modern software relies on hundreds of APIs"
  - "One misuse can compromise entire systems"

- **Transition**: "So why do current solutions fail?"

Slide 5: Why Current Solutions Fall Short (1.5 minutes)

- **Table walkthrough**:
  - Documentation: "64 pages just for Java crypto - who reads this?"
  - Static analysis: "67 false positives in our study - alarm fatigue"
  - External tools: "Developers already juggling 10+ tools"
- **Core problem highlight**:
  - "Same API, different contexts"
  - "Android vs JDK, BSI vs NIST standards"
  - "Version changes break assumptions"
- **Key insight**: "Context matters, but tools ignore it"
- **Transition**: "Let's see the specific patterns..."

Slide 6: Three Main API Misuse Patterns (Simple Version) (30 seconds)

- **Quick overview**: Point to each box
- **MuBench reference**: "Based on real GitHub projects"
- **Transition quickly**: "Let me show you these in detail..."

Slide 7: Three Main API Misuse Patterns (Detailed) (1.5 minutes)

- **Triangle layout explanation**:
  - Top left - ECB: "Like using transparent envelopes for secrets"
  - Top right - Sequence: "Like signing a blank check"
  - Bottom - Composition: "Like a paper lock on a bank vault"
- **Visual impact**: Let the images speak
- **Key insight at bottom**: "All involve state machines - this is crucial"
- **Transition**: "APIs have hidden state machines..."

Slide 8: API Usage as State Machines (1 minute)

- **Core insight**: "APIs aren't just functions - they're state machines"
- **Visual explanation**: Point to state transitions
- **Problem**: "This state is implicit, hidden in docs"
- **DSL advantage**: "What if we made it explicit?"

- **Transition**: "CrySL tried this approach..."

Slide 9: DSLs to the Rescue - CrySL (1.5 minutes)

- **Quick intro**: "CrySL specifies correct patterns externally"
- **Two columns**:
  - Left: "Structure and events"
  - Right: "Constraints and order"
- **Note improvements**: "Proper syntax highlighting for readability"
- **944 rules**: "Comprehensive but..."
- **Limitations**:
  - "Separate from code"
  - "Needs external tools"
  - "No runtime state"
- **Transition**: "What if specs lived IN the API?"

Slide 10: The Challenge (30 seconds)

- **Quick recap**: "Traditional approaches have fundamental limitations"
- **Set up jGuard**: "We need a new paradigm..."

Slide 11: JGuard: A New Approach (1 minute)

- **Paradigm shift**: "APIs protect themselves"
- **Car analogy**: "Won't start without seatbelt"
- **Flow walkthrough**: Each step builds on previous
- **Key advantage**: "Zero new tools for users!"
- **Implementation note**: "Built with JetBrains MPS"

Slide 12: JGuard: Making State Machines Explicit (1.5 minutes)

- **Two-part slide**:
  - Left: "Conceptual flow of state tracking"
  - Right: "Actual code implementation"
- **State machine encoding**: Walk through vertical flow
- **Code example**: "Guards make state explicit"
- **Key innovation**: "State machines as first-class citizens"

- **Transition**: "Let's see the transformation..."

Slides 13-17: JGuard Technical Details (5 minutes total)

Quick pace through technical slides - focus on transformation aspect

Slide 13: Guards (1 minute)

- **Show transformation**: "DSL → Java"
- **Key point**: "Simple boolean fields with finalizer checks"

Slide 14: Requirements (1 minute)

- **Transformation focus**: "Requirements become runtime checks"
- **Order matters**: "Null check first for clear errors"

Slide 15: Generated Checks (30 seconds)

- **Quick point**: "Clean, efficient generated code"

Slide 16: Consequences (1 minute)

- **Iterator example**: "Everyone knows this pain"
- **Wrapper pattern**: "Intercepts returns"

Slide 17: Exception Handling (30 seconds)

- **Often missed**: "State consistency even on failure"

Slide 18: Visualizing State Transitions (45 seconds)

- **Success path**: Quick walkthrough
- **Failure path**: "Common mistake caught"
- **Visual impact**: Let diagram speak

Slide 19: Meta Variables (1 minute)

- **Context problem**: "Different standards, same API"
- **Solution**: "Compile-time specialization"
- **Practical**: "Ship different JARs for different contexts"

Slide 20: Empirical Validation (1.5 minutes)

- **Three metrics at once**:

    - Expressiveness: "89.2% - matches real misuses"
    - Accuracy: "ZERO false positives!"
    - Performance: "Negligible overhead"
- **Let numbers sink in**: Especially zero false positives
- **Deployment strategy**: "Dev/test on, production off"
- **Transition**: "From specific to general..."

Slide 21: From Domain-Specific to Broader Challenges (1 minute)

- **Acknowledge success**: "jGuard works for APIs"
- **Broader view**: "But developers face many challenges"
- **Bridge concepts**: Walk through flow diagram
- **Model First, AI Next**: "This philosophy is key"
- **Transition**: "LLMs promise to help, but..."

Slide 22: Promise of LLMs (45 seconds)

- **Current capabilities**: Quick mention
- **SE 3.0 vision**: "Natural language as primary interface"
- **But...**: "There's a fundamental problem..."

Slide 23: The Black Box Problem (1 minute)

- **List limitations**: Each is serious
- **Visual flow**: "Plausible but dangerous"
- **Security risks**: "Our studies show vulnerabilities"
- **Transition**: "Recent research reveals why..."

Slide 24: Why LLMs Fail - Dictionary Connections (2 minutes)

CRITICAL SLIDE - Take time here

- **Research citation**: "Anand et al. 2024 - groundbreaking findings"
- **Dictionary problem explanation**:
    - "Syntax matches syntax ✓"
    - "Variables match variables ✓"
    - "But syntax CANNOT connect to meaning ✗"

- **Visual flow**: Walk through each node slowly
- **Counterintuitive findings**:
  - "Larger models WORSE!"
  - "Fine-tuning makes it WORSE!"
- **Solution path**: "Domain models provide missing bridge"

Slide 25: Key Findings (45 seconds)

- **Reinforce previous slide**: "This explains everything"
- **Expert compensation**: "Humans provide semantic bridge"
- **Our approach**: "Make that bridge explicit"

Slide 26: Expertise Gap (1 minute)

- **TOMMY study**: Visual contrast clear
- **Expert vs novice**: Point to differences
- **Technical gaps**: Both critical
- **Transition**: "This motivates our solution..."

Slide 27: Our Vision - Combined (1.5 minutes)

- **Full width impact**: "Using all screen space now"
- **Flow explanation**: Each component addresses a gap
- **DSL components**: Two complementary models
- **Code example**: "Formal representation enables reasoning"
- **Key point**: "Context makes AI trustworthy"

Slide 28: Developer Context DSL (Skip - already covered)

Slide 29: Case Study - ECB (2 minutes)

- **Full width layout**: "Better visibility"
- **Problem**: "Pattern matching leads to ECB"
- **Flow**: Walk through each step
- **Before/after**: "Clear improvement"
- **Personalized explanation**: "Adapts to developer level"
- **Impact**: "Combines fluency with correctness"

Slide 30: Iterative Refinement (45 seconds)

- **Continuous improvement**: Quick cycle explanation
- **Trust but verify**: "Best of both worlds"
- **Transition**: "Future opportunities..."

Slide 31: Future Directions (30 seconds)

- **Quick overview**: "Each challenge is an opportunity"
- **Collaboration hint**: "Speaking of opportunities..."

Slide 32: Research Validation (1 minute)

- **Three pillars**: Each supports our approach
- **Recent citations**: "All 2024-2025 work"
- **Our contribution**: "First to combine all three"
- **Transition**: "Ready for conclusions..."

Slide 33: Conclusion (1.5 minutes)

- **Journey recap**:
  - "Started with API safety"
  - "Discovered why LLMs fail"
  - "Presented bridging solution"
- **Key contributions**: Hit all three
- **Vision**: "Trustworthy AI-native SE"
- **Thank you**: "I look forward to your questions"

Collaboration Slides (1-2 minutes each, if time permits)

Present 2-3 based on audience composition

General approach:

- Start with their recent work
- Show specific synergy
- Propose concrete project
- Emphasize mutual benefit

Time Management

- Introduction & API Problem: 10 minutes
- jGuard Technical: 7 minutes
- LLM Vision & Integration: 10 minutes
- Conclusion: 2 minutes
- Collaboration (optional): 3-6 minutes
- **Total: 29-32 minutes**

Critical Reminders

1. **Projector visibility**: Code is now enhanced with stronger colors and larger fonts
2. **Combined slides**: Use full width, especially for case study
3. **Dictionary connection**: This is THE key insight - spend time here
4. **Collaboration**: Only if time permits, but be ready with 2-3

Q&A Preparation

Expected Questions:

1. "How does jGuard compare to contracts/assertions?"
   - More expressive (state machines)
   - Context-aware (meta variables)
   - Zero false positives
2. "What about performance overhead?"
   - Show empirical results
   - Deployment flexibility
   - Compare to assertion overhead
3. "Why not just better documentation?"
   - 64 pages for one API
   - Developers don't read
   - Context variations
4. "How does this scale?"
   - Compositional approach
   - Per-API specifications
   - Automated generation possible

5. "What about legacy code?"
    - Gradual adoption
    - Wrapper approach
    - Tool support for migration

Final Tips

- **Energy**: Project excitement about Leiden specifically
- **Concrete examples**: Use real code and impacts
- **Visual aids**: Let enhanced diagrams speak
- **Forward-looking**: End each section with what's next
- **Collaboration**: Make it about THEM, not you

Good luck! You've got this! 🎯