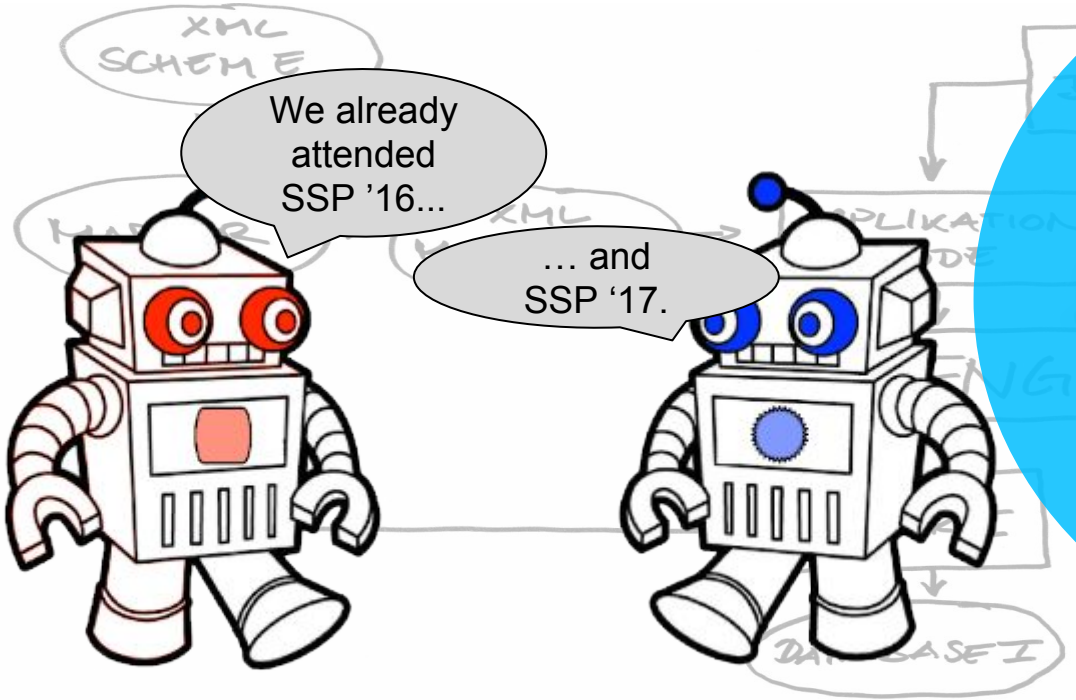
University of Stuttgart
Institute of Software Technology
Reliable Software Systems

Universidad Nacional del Centro (UNICEN)
ISISTAN Research Institute
Software Engineering Group



Using the CoCoME community case study for evaluating the SQuAT architecture optimization approach: results and experiences

Sebastian Frank
Santiago Vidal

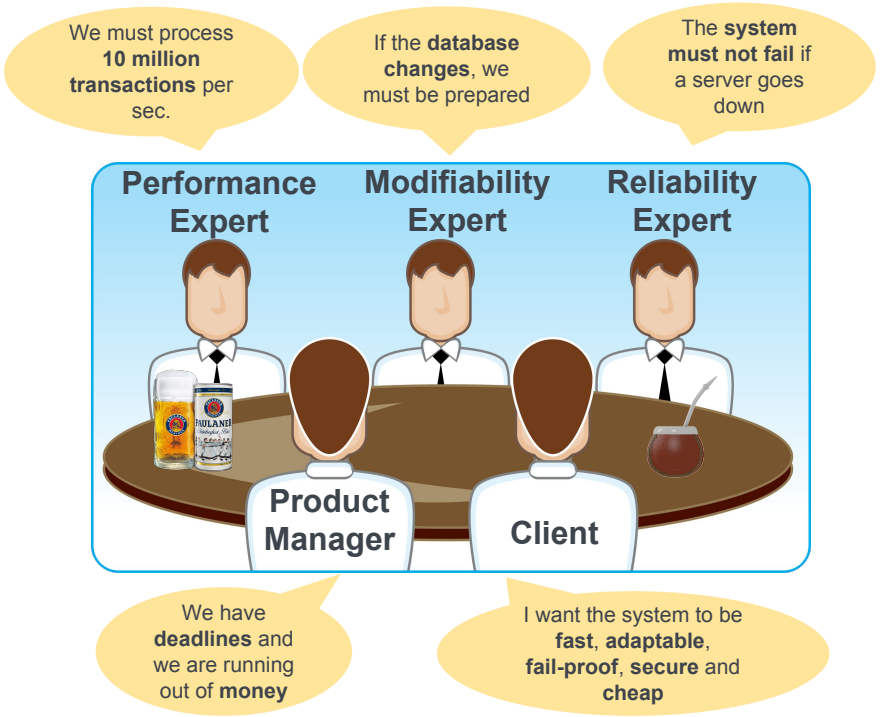
GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

RSS Reliable Software Systems
ISISTAN Instituto Superior de Ingeniería de Software Tandil
Design For FUTURE
DFG
DFG Priority Programme 1593
Design For Future - Managed Software Evolution

SQuAT Approach

Motivation

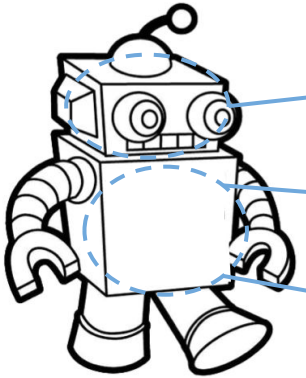
- **Difficulties** to manage and optimize **all QAs** in a satisfying way, i.e., make everybody happy
- Monolithic analysis is **unnatural**
- Trade-offs between different QA goals & **negotiation among experts** in software architecture



SQuAT Approach

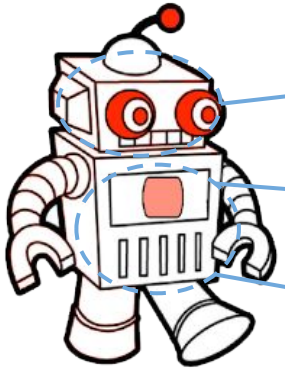
The Bots

Abstract Bot



- Scenario (Goal)
- Model Analyzer
- Model Optimizer

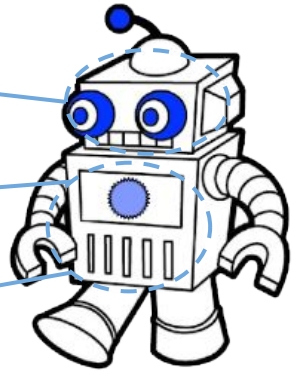
Performance Bot



- Workload, Usage Behaviour, CPU Clock Rate
- Headless Palladio LQN Solver
- Headless PerOpteryx (Evolutionary Algorithm)

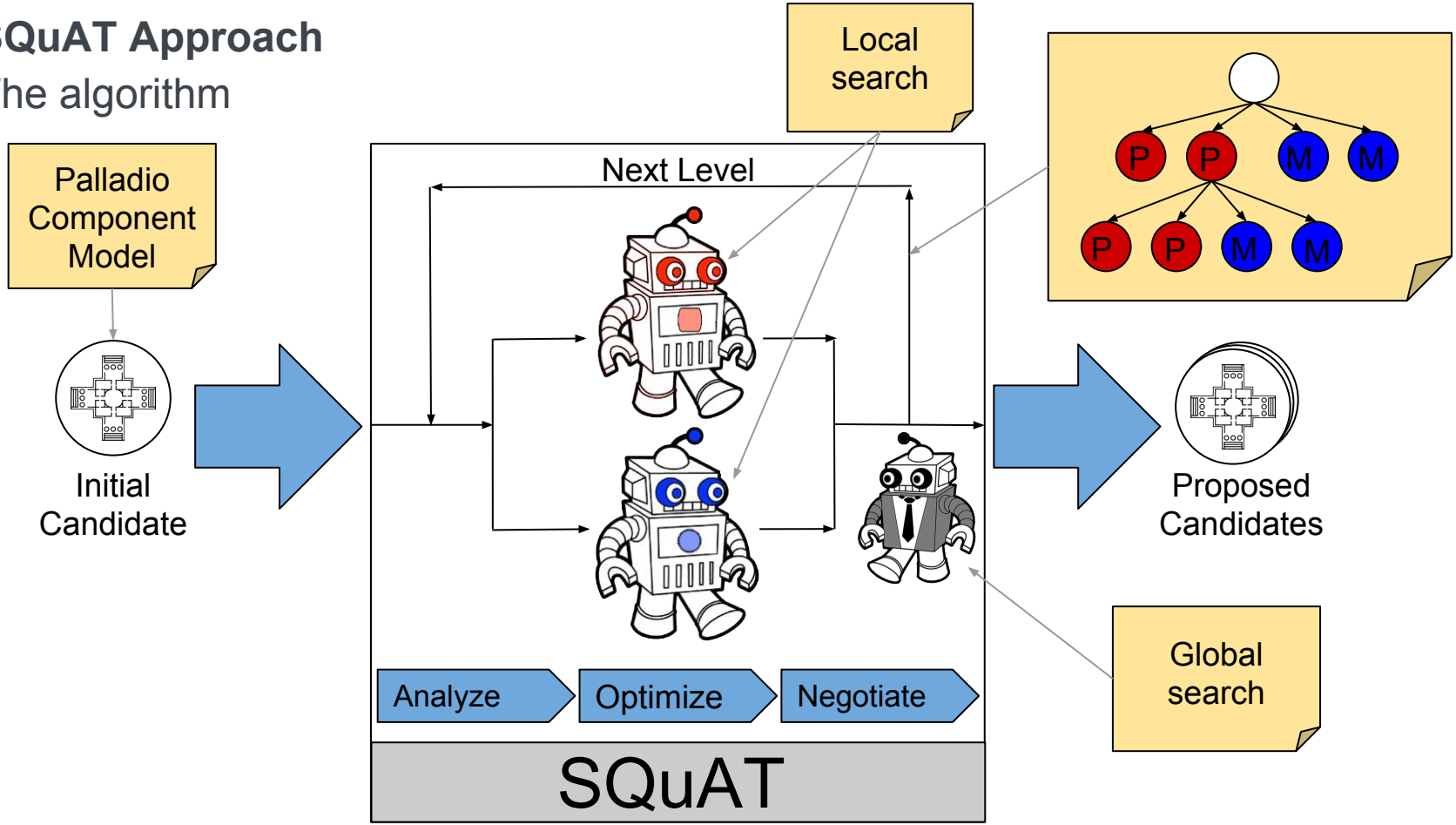
Modifiability Bot

- Add, Remove, Modify Components & Interfaces
- KAMP Extension (Change Impact Analysis)
- Henshin Rules



SQuAT Approach

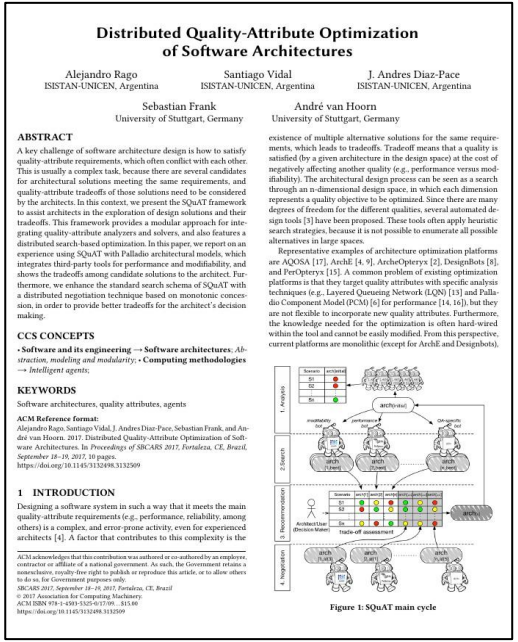
The algorithm



Case Study

CoCoME

See **SSP '17** and/or
"Distributed Quality-Attribute Optimization of Software Architectures" (SBCARS)



Distributed Quality-Attribute Optimization of Software Architectures

Alejandro Rago, ISISTAN-UNICEN, Argentina
Santiago Vidal, ISISTAN-UNICEN, Argentina
J. Andres Diaz-Pace, ISISTAN-UNICEN, Argentina
Sebastian Frank, University of Stuttgart, Germany
André van Hoorn, University of Stuttgart, Germany

ABSTRACT

A key challenge of software architecture design is how to satisfy quality-attribute requirements, which often conflict with each other. This is usually a complex task, because there are several candidates for architectural solutions meeting the same requirements, and quality-attribute tradeoffs of those solutions need to be considered by the architects. In this context, we present the SQuAT framework to assist architects in the exploration of design solutions and their tradeoffs. This framework provides a modular approach for integrating quality-attribute analyzers and solvers, and also features a distributed search-based optimization. In this paper, we report on an experience using SQuAT with Palladio architectural models, which integrates third-party tools for performance and modifiability, and shows the tradeoffs among candidate solutions to the architect. Furthermore, we enhance the standard search schema of SQuAT with a distributed negotiation technique based on monotonic concession, in order to provide better tradeoffs for the architect's decision making.

CCS CONCEPTS

• **Software and its engineering** → **Software architectures**; *Abstraction, modeling and modularity*; • **Computing methodologies** → *Intelligent agents*;

KEYWORDS

Software architectures, quality attributes, agents

ACM Reference format:
Alejandro Rago, Santiago Vidal, J. Andres Diaz-Pace, Sebastian Frank, and André van Hoorn. 2017. Distributed Quality-Attribute Optimization of Software Architectures. In *Proceedings of SBCARS 2017, Fortaleza, CE, Brazil, September 18–19, 2017*, 10 pages.
https://doi.org/10.1145/3132498.3132509

1 INTRODUCTION

Designing a software system in such a way that it meets the main quality-attribute requirements (e.g., performance, reliability, among others) is a complex, and error-prone activity, even for experienced architects [4]. A factor that contributes to this complexity is the existence of multiple alternative solutions for the same requirements, which leads to tradeoffs. Tradeoff means that a quality is satisfied (by a given architecture in the design space) at the cost of negatively affecting another quality (e.g., performance versus modifiability). The architectural design process can be seen as a search through an n-dimensional design space, in which each dimension represents a quality objective to be optimized. Since there are many degrees of freedom for the different qualities, several automated design tools [3] have been proposed. These tools often apply heuristic search strategies, because it is not possible to enumerate all possible alternatives in large spaces.

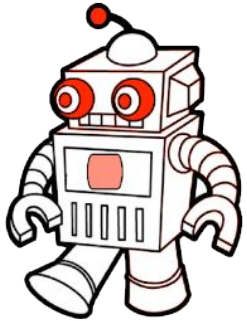
Representative examples of architecture optimization platforms are AQOSA [17], ArchE [4, 9], ArcheOpteryx [2], DesignBots [8], and PerOpteryx [15]. A common problem of existing optimization platforms is that they target quality attributes with specific analysis techniques (e.g., Layered Queueing Network (LQN) [13] and Palladio Component Model (PCM) [6] for performance [14, 16]), but they are not flexible to incorporate new quality attributes. Furthermore, the knowledge needed for the optimization is often hard-wired within the tool and cannot be easily modified. From this perspective, current platforms are monolithic (except for ArchE and Designbots),

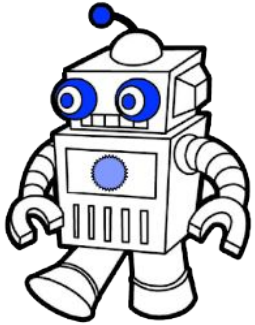
Figure 1: SQuAT main cycle

	Case Study: ST+	Case Study: CoCoME
About	Business Trip Management	Trading System (supermarket)
Size	**5** Components **4** Alternative Components	**51** Components **5** Alternative Components
Scenarios	**2** Performance Scenarios **2** Modifiability Scenarios	**4** Performance Scenarios **4** Modifiability Scenarios
Levels	**2**	**5**
Quality	Fully Modeled "**Clean**"	Partially Modeled (~25%) "**Dirty**"

Case Study

Scenarios in the CoCoME Case Study



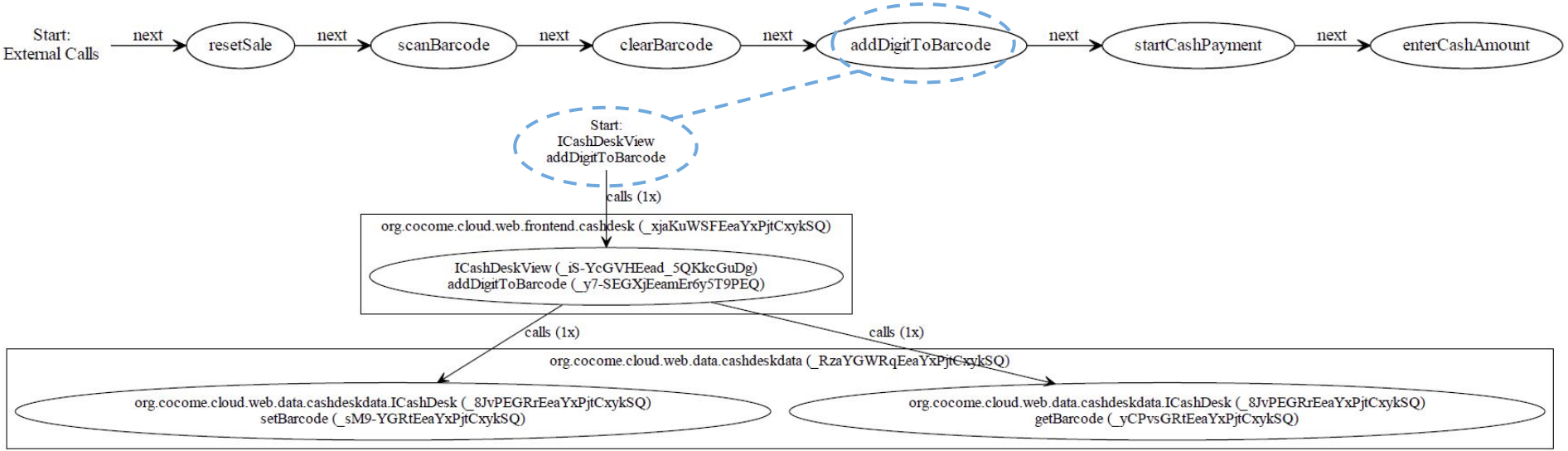


Name	Stimulus	Expected Response
P1	Overall Usage Increase +10%	< 1.2 seconds
P2	Overall Usage Increase +50%	< 1.4 seconds
P3	Server Fault (1 out of 2 in a cluster)	< 1.0 seconds
P4	Usage Increase of Specific Service	< 2.4 seconds
M1	Add NFC Payment Method	< 2270 (complexity)
M2	Add Premium User Service	< 750 (complexity)
M3	Add Withdraw Money Function	< 170 (complexity)
M4	Add Logging Function	< 2180 (complexity)

Experiences

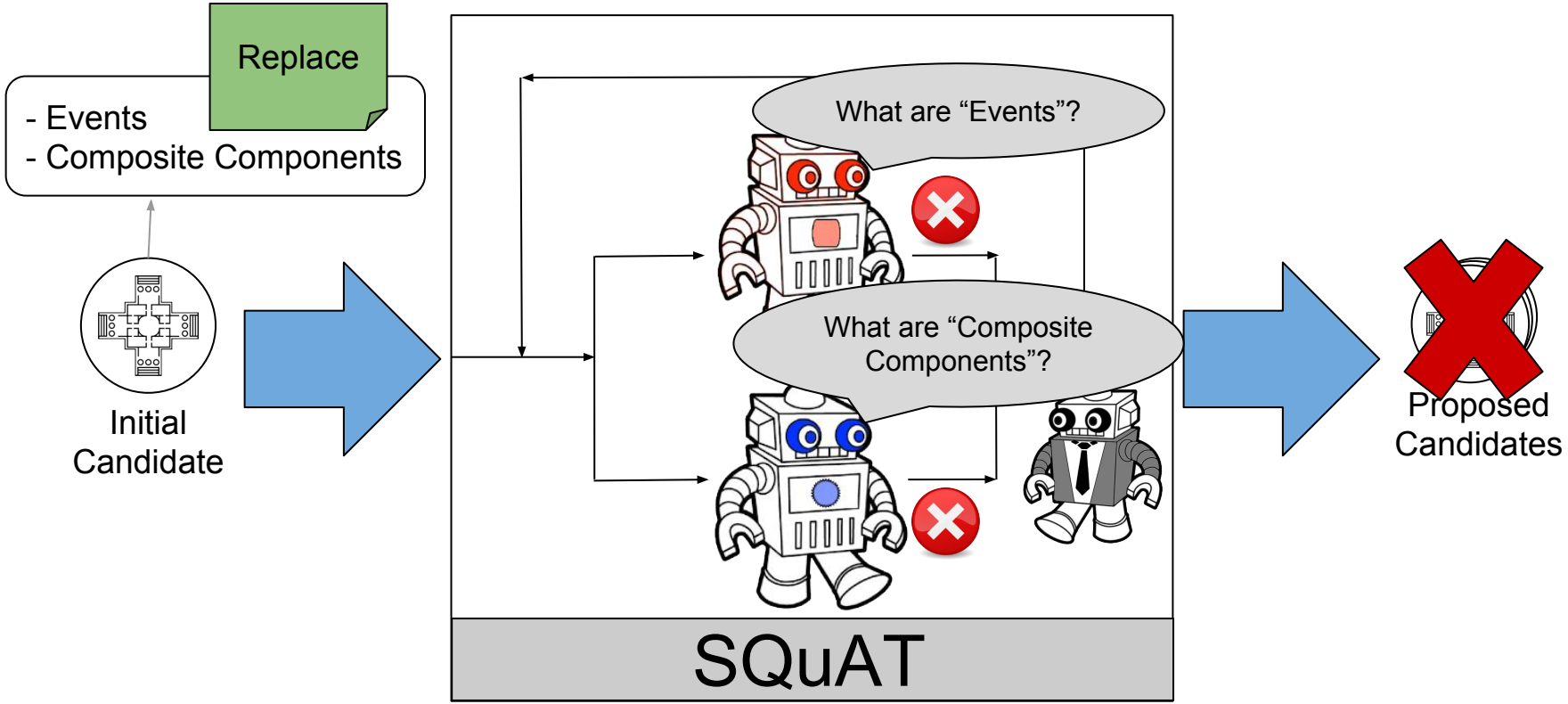
Model: Preparations

- Model Allocation, Usage Behaviour,...
- Understanding CoCoME
  - Call Graph Tool
    - Side Effect: Bug Fixing



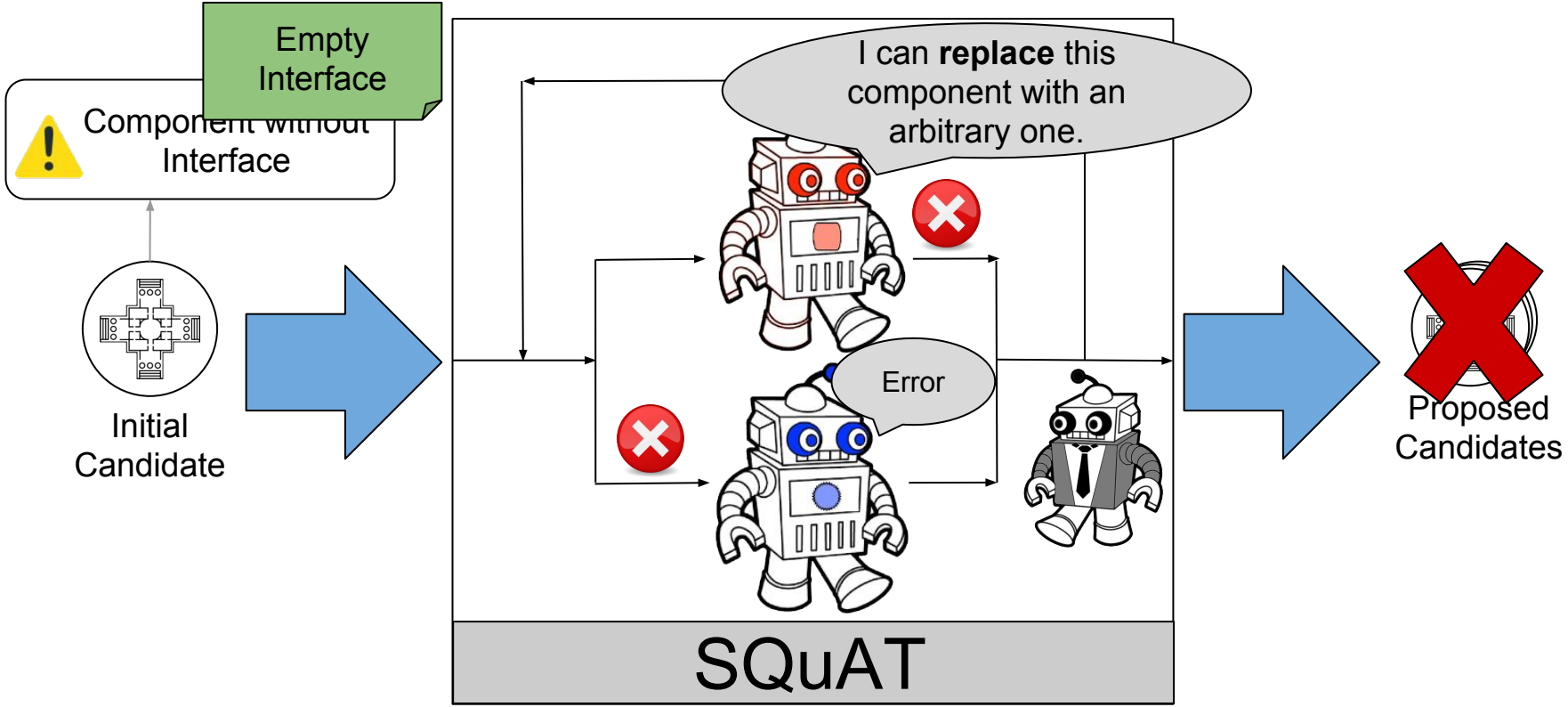
Experiences

Model: Limitations



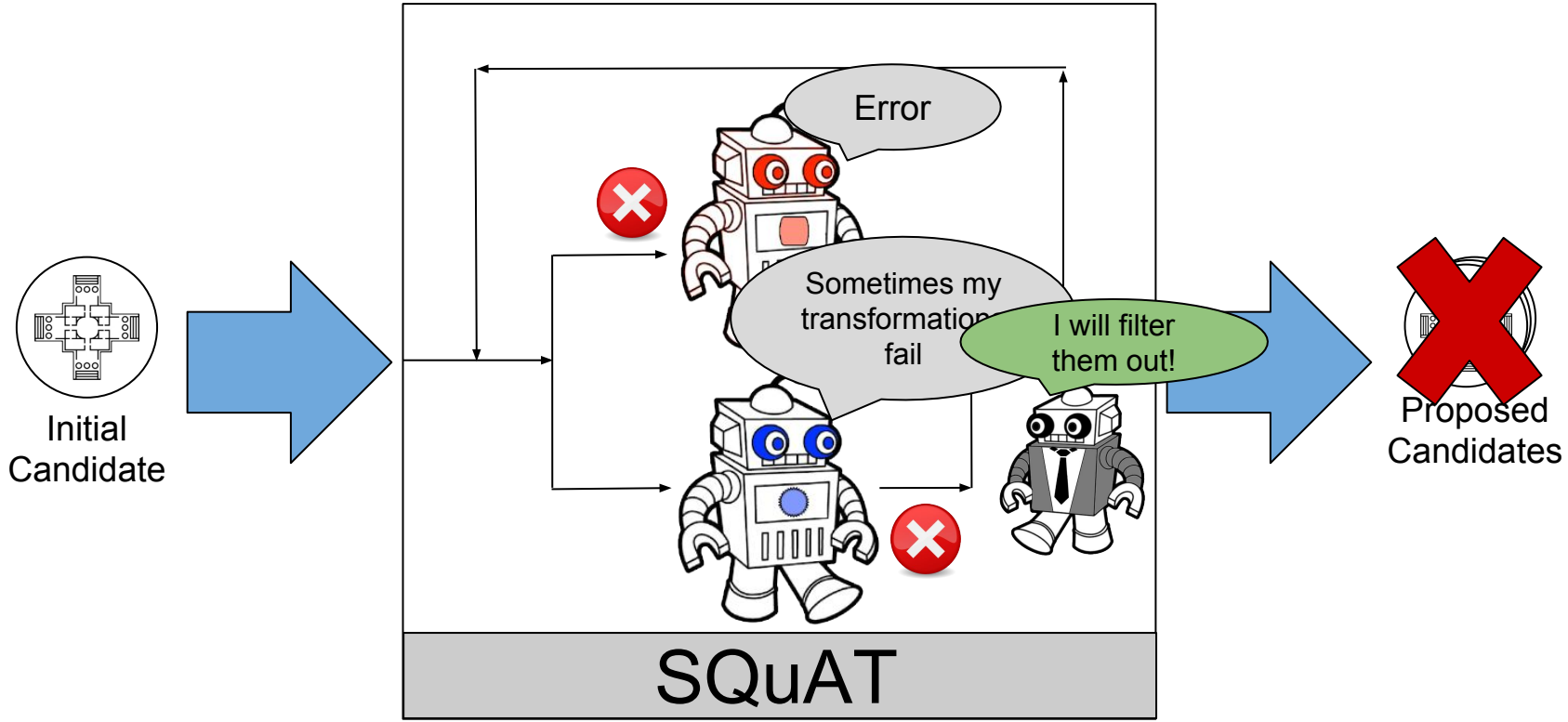
Experiences

Model: Bad Practices

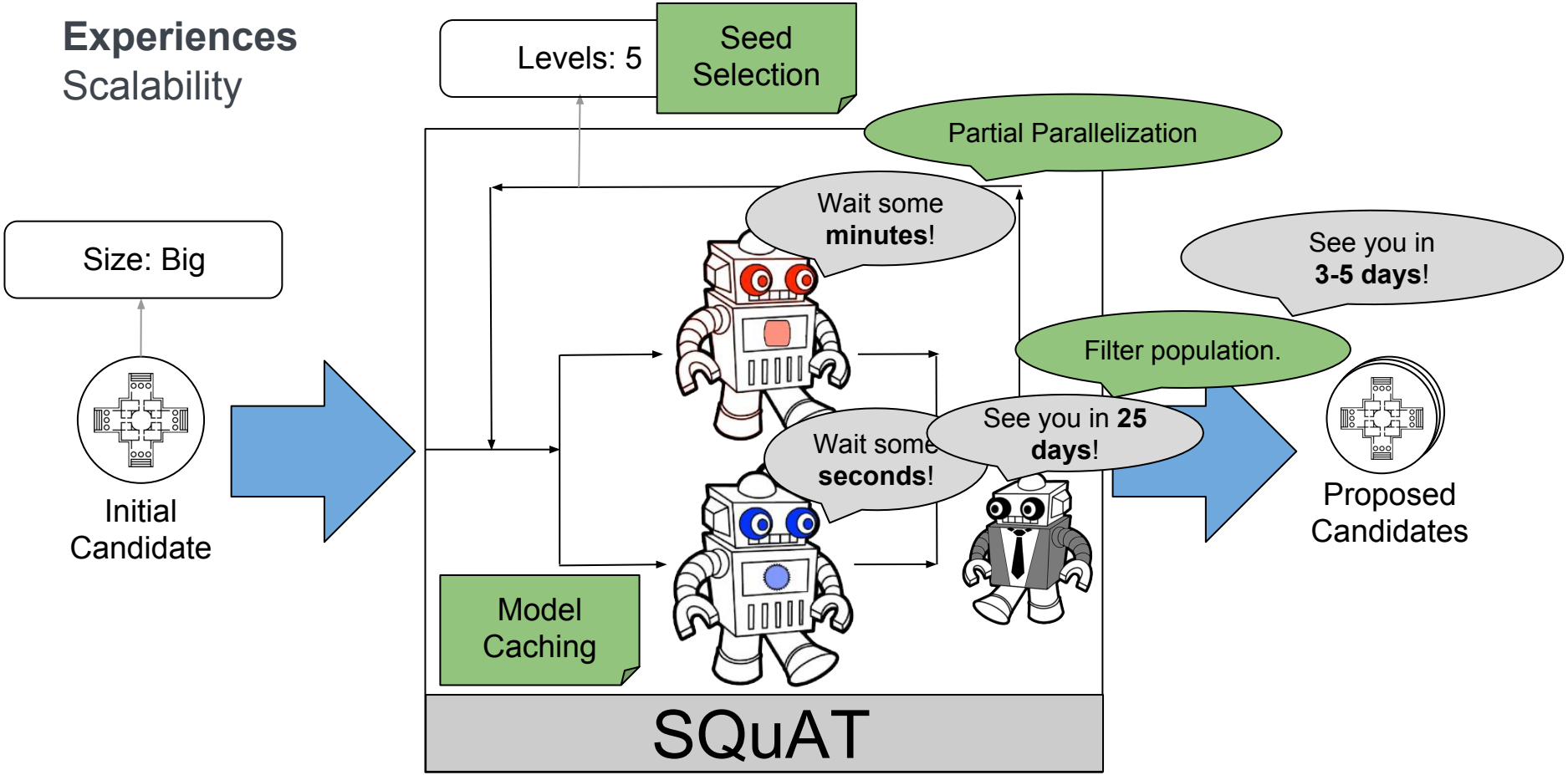


Experiences

Model: Validity

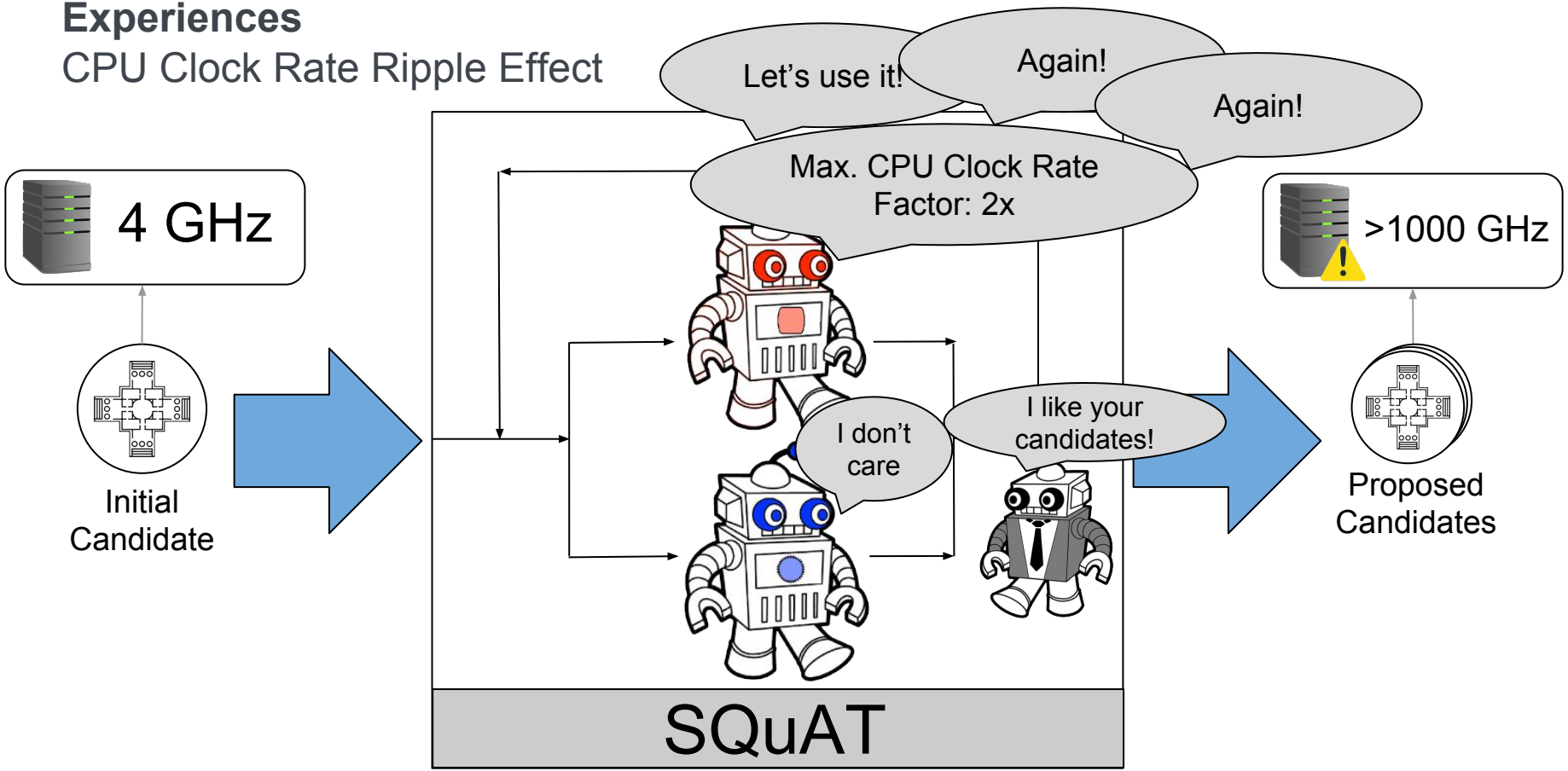


Experiences
Scalability



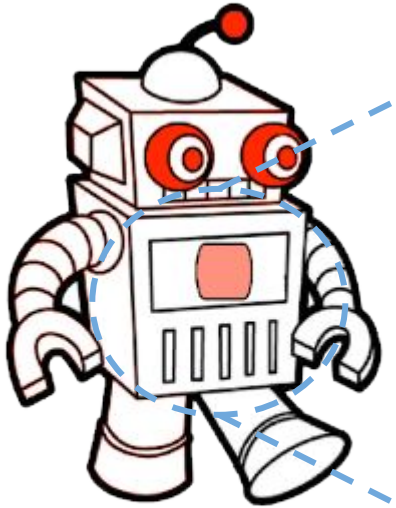
Experiences

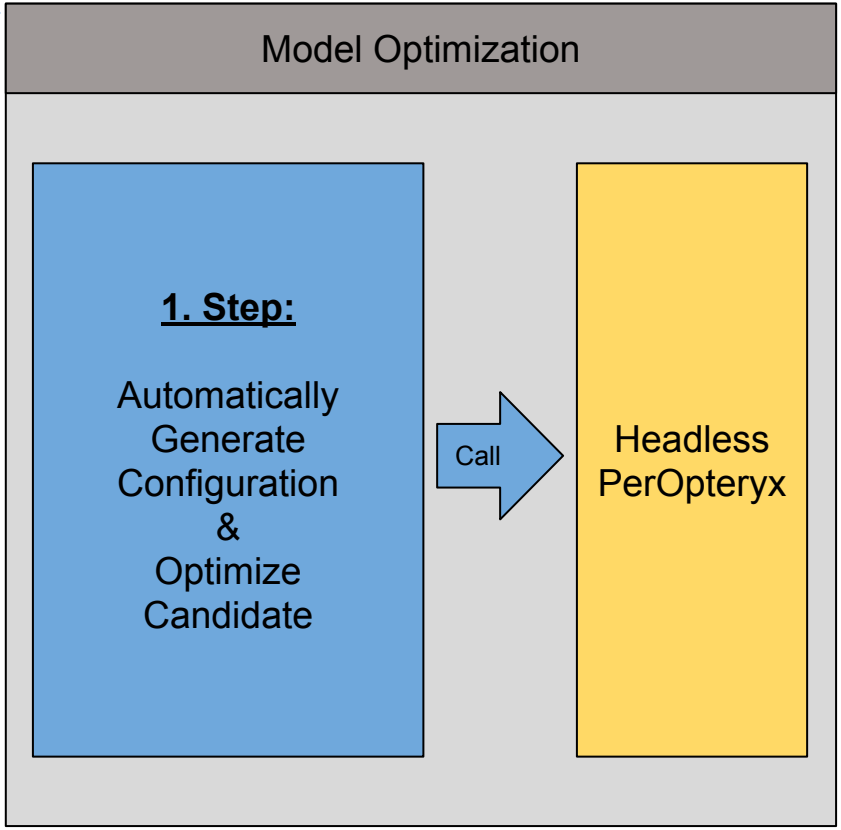
CPU Clock Rate Ripple Effect



Experiences

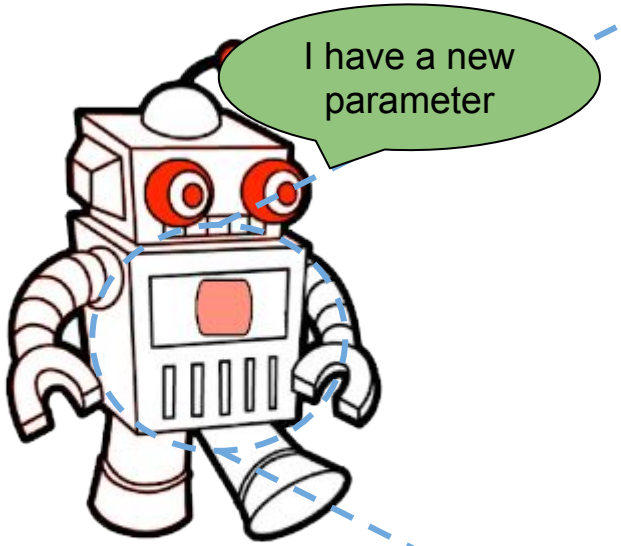
CPU Clock Rate Ripple Effect

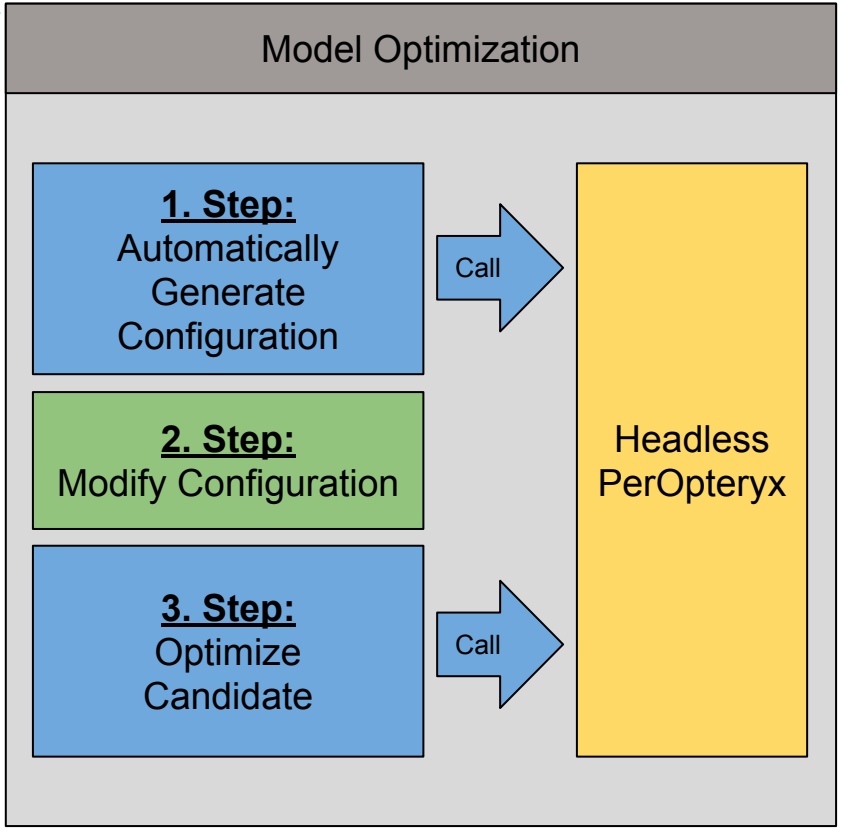




Experiences

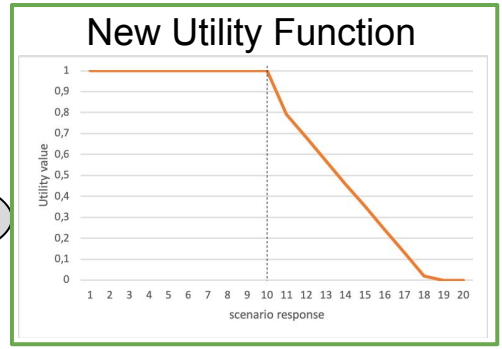
CPU Clock Rate Ripple Effect

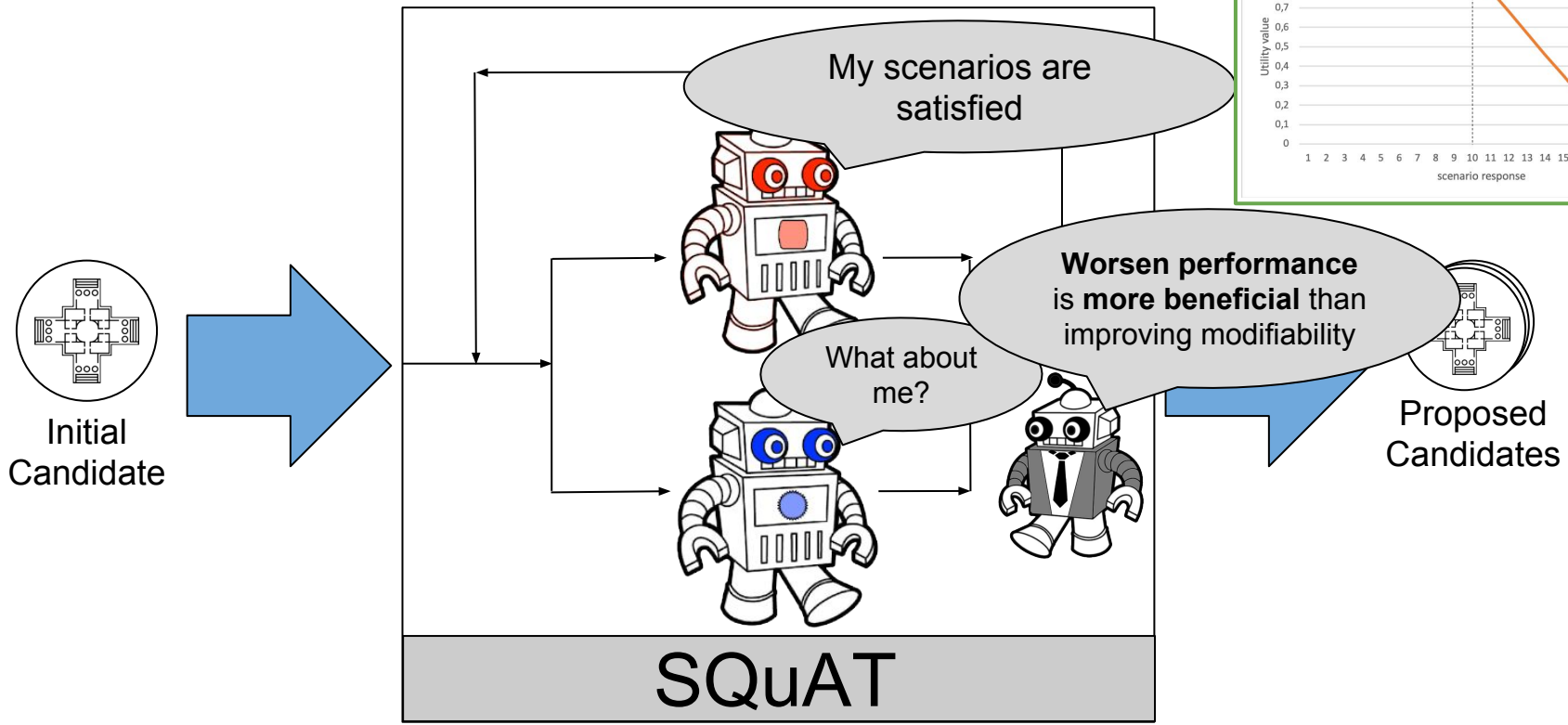




Experiences

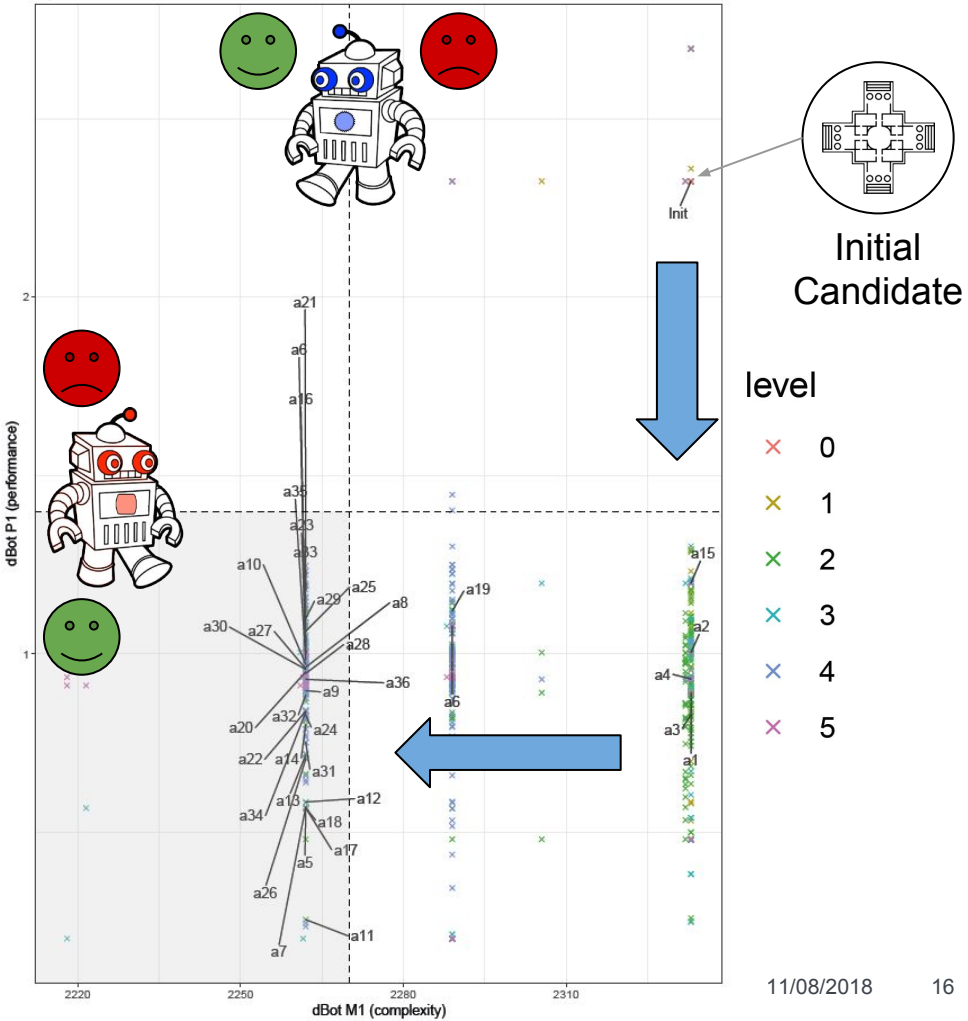
Utility Function





Results
M1 - P1

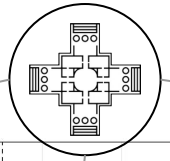
- Many “good” agreements
  - satisfy Performance Scenario 1
  - satisfy Modifiability Scenario 1
- Higher levels are useful

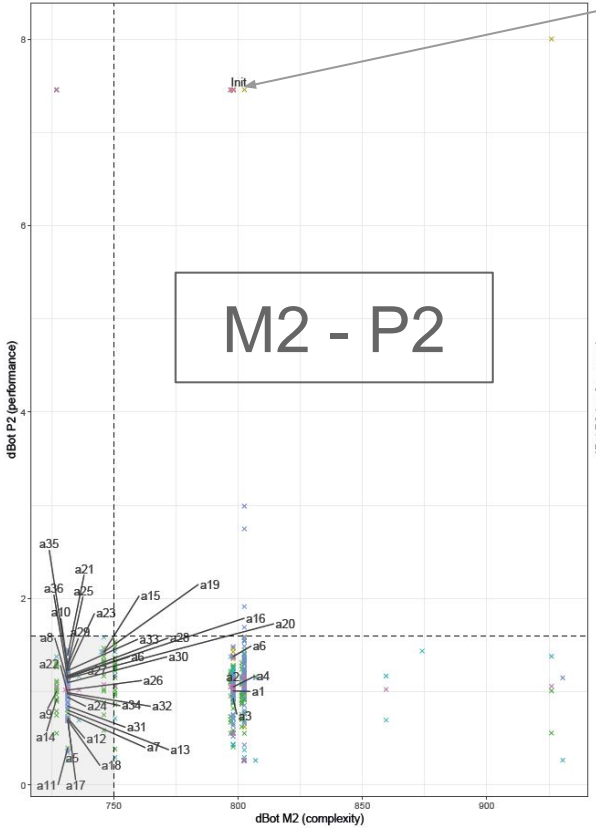


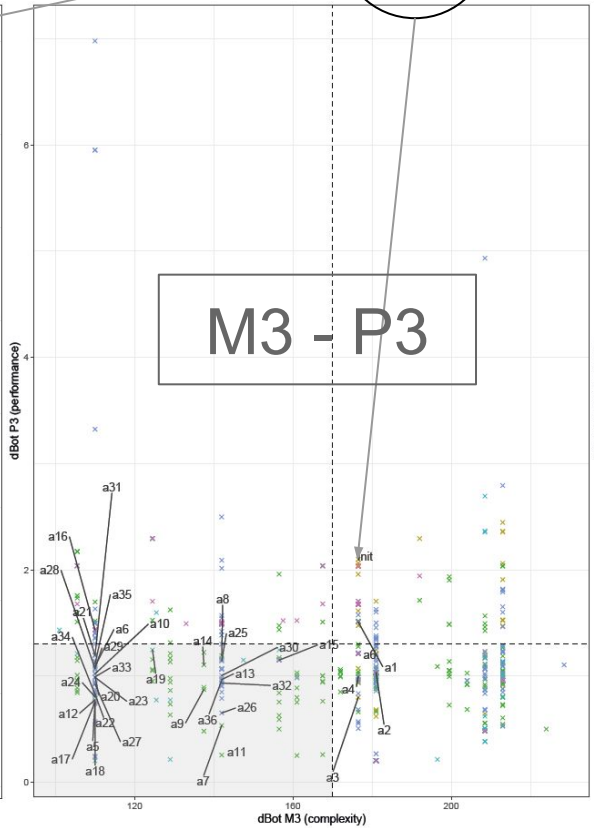
Results

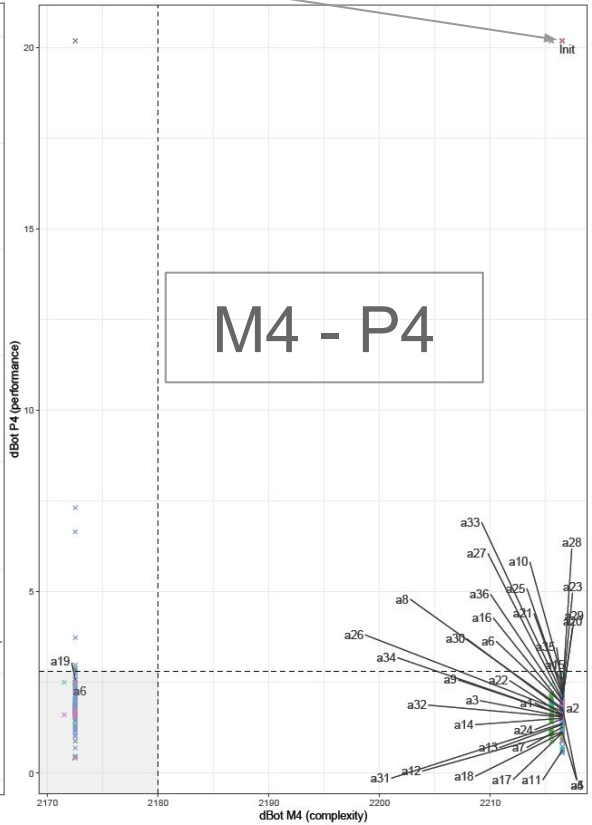
More Results

Initial Candidate









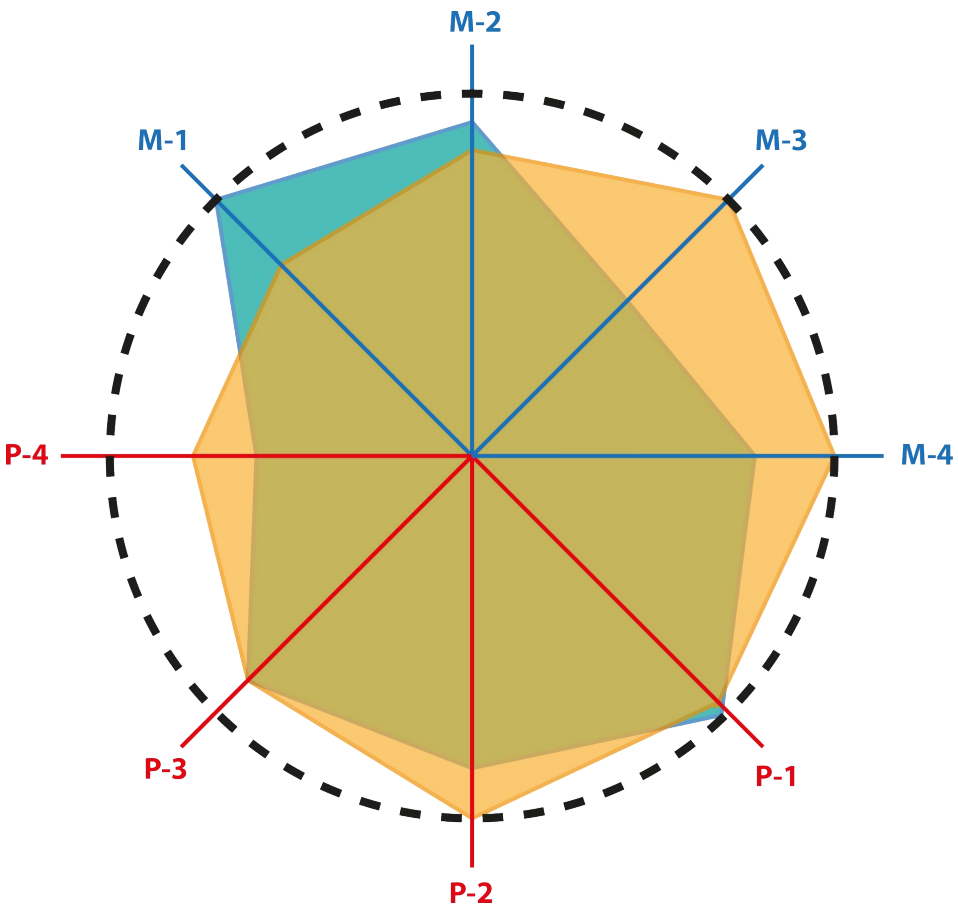
level

- 0
- 1
- 2
- 3
- 4
- 5

Future Work

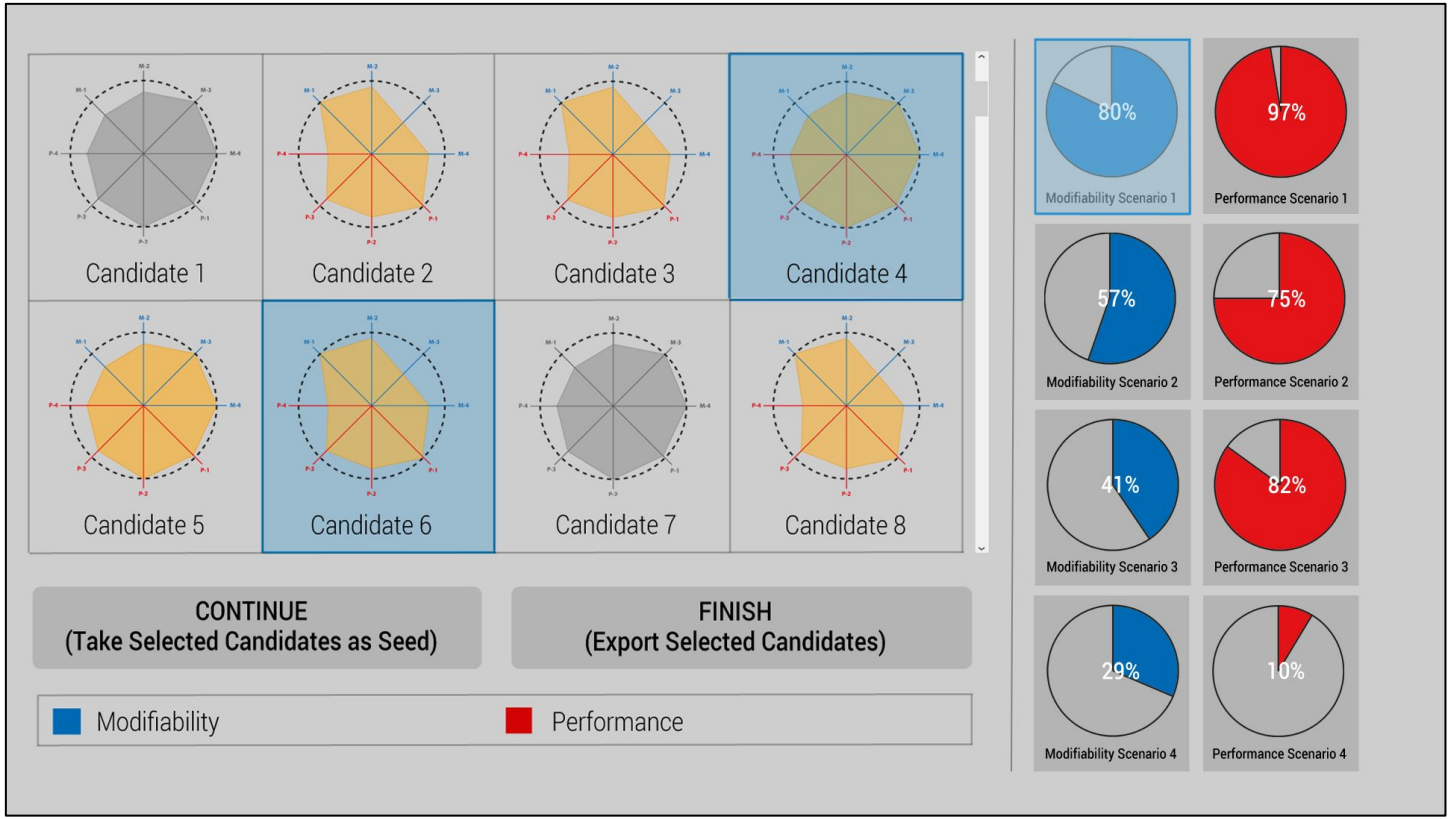
Lessons Learned

- Model
  - Quality (“Bad Practices”)
  - Exception Handling Strategies
    - Filters, ...
- Visualization
  - Understand Changes
  - Compare Models
- Interaction
  - New Insights at Runtime
  - Human Knowledge



Future Work

Visualization & Interaction



Future Work

More Ideas

- Personalization of Bots
  - Own Utility Function, ...
- Scalability
  - Microservices
  - Further Parallelization, Caching, ...
- Usability
  - Natural Language
    - Scenarios & Results
  - Machine Learning