Afstuderen bij Axini

Houd je van breinkracers, onderzoek vertalen naar de praktijk?
Ben je niet bang voor techniek, software development en toolontwikkeling?
Heb je zin om te werken in een klein, gespecialiseerd team?

Axini is een spin-off van de Universiteit Twente. Technologie en theorie die vers uit de onderzoeks-wereld komen passen we toe op complexe systemen van klanten. Daarbij ontwikkelen we onze eigen toolset. Dit biedt mooie afstudeer-mogelijkheden.

Wat zijn jullie voor bedrijf?
Axini is voortgekomen uit promotie-onderzoek naar model-gebaseerd testen. We hebben dit doorontwikkeld tot tooling waarmee de meest complexe systemen volledig automatisch en grondiger dan ooit kunnen worden getest. Dit is uniek en erg succesvol. We hebben voornamelijk klanten: banken, verzekeraars, logistieke partijen, fabrikan ten van navigatiesystemen, betaalsystemen tot medische. We groeien daarom flink maar zijn, met acht enthousiaste technneuten, nog prettig klein en informeel.

Wat is model-gebaseerd testen?
Testen van software bestaat uit drie stappen: bedenken van testen, uitvoeren van testen en het controleren van de testuitkomst. Model-gebaseerd testen kan deze drie stappen automatiseren. De functionaliteit van het te testen systeem wordt beschreven in een van onze domeinspecifieke modellerte- talen. Uit dit model kan TestManager, ons modelleren en test-tool, gericht testen afleiden, uitvoeren op het te testen systeem en vervolgens controleren. Daarna volgt analyse: hoe goed is er nu getest? Wat zijn mogelijk onderliggende foutoorzaken?

Waarom zou ik bij jullie afstuderen?
We hebben legio onderwerpen waar je relevant onderzoek naar kan doen. Van theoretisch tot praktisch, van constraint solving tot datavisualisatie, met veel ruimte voor eigen invulling. We bieden goede, inhoudelijke begeleiding en vaak is er de mogelijkheid om je werk in de praktijk te toetsen bij een van onze klanten. We werken met allerhande hippe open source spullen. Er valt dus het nodige te beleven en te leren. En last but not least: ons kantoor is in Amsterdam, tussen de bakker en de kroeg.

Welke studenten zoeken jullie?
We zoeken slimme, gedreven studenten die van hun vak houden en er plezier in hebben om de gebaande paden te verlaten en nieuwe wegen te ontdekken. Veranderen van taal of tools deert je niet en je haalt voldoening uit het leveren van mooi werk.

Contact
Je vindt ons op Azartplein 12, Amsterdam. Bel voor meer informatie Menno Jonkers op (06) 2476 4265, of mail info@axini.com.
Examples of potential graduation projects

In addition to the examples listed here, there are many other options in our broad field of research. Tell us what you're looking for and we can check if there's a fit.

Example 1: Model Learning

Problem
In order to test if a system is working correctly, we need a model of its behavior. When a system is already implemented, we could use a method that can automatically obtain such a model. Unfortunately, no such method exists yet for real life situations.

To be researched
Develop a learning technique to learn a model that can be used for model-based testing by observing its behavior and evaluate it. Challenges are non-deterministic systems, systems with concurrent behavior and systems that deal with complex data.

Evaluation
The learning method will be used to obtain models for systems of clients of Axini in order to lower the threshold of starting with model-based testing.

Context
Axini uses Symbolic Transition Systems for modeling. Most existing learning techniques use other forms of modeling and do not support the complexity of practical usage. This project can be done in collaboration with the ITALIA project of the University of Nijmegen. ITALIA is a big research program where other PhD students and Master students are working on model-learning puzzles http://www.italia.cs.ru.nl/

Example 2: Model-checking

Problem
Model-based testing involves making a (formal) model. We make these models by hand. Because most of the systems we model are complex, the models tend to become big and complex as well. This makes it hard for people to understand them and to verify whether they are correct. There is an entire research area called model-checking that is about properties and correctness of models. For our modeling formalism: Symbolic Transition Systems, there is no model-checker. This assignment is about making a model-checker for Symbolic Transition Systems. Efficiency is not important (for a start), correctness is. This is new work, because there is no existing checker for symbolic models, and it is interesting because of the famous state-space explosion problem.

This project is a collaboration with Jan van Eijck (UvA)

To be researched
The formal models we use for modeling are Symbolic Transition Systems. These models are very powerful, because they can model systems with parallel and data behavior very well. One of the challenges of making a model checker will be to overcome the famous state-space explosion problem these kind of systems usually bring along. Another challenge will be to translate the theory on normal labeled transition systems to our symbolic labeled transition systems.
Evaluation
The model checker (prototype) built can be used immediately in one of the active projects, so we can immediately evaluate the added value.

Context
An STS or Symbolic Transition System consists of:
- a name
- a set of variables
- a set of labels, called gates in the attached theory
- a set of states, called locations in the attached theory
- a set of transitions, called switches in the attached theory
- a set of start states
- an initial valuation.

This is the static definition of the STS. When exploring an STS, variables get values, this means that there is also a Global Valuation. This is a mapping of variable-names to values. The way variables get updated is via the execution of transitions.

Of prime interest (and some confusion probably) is the transition. A transition has:
- a source state and a target state
- a guard. This is a logical formula over some variables, evaluating to true or false.

Whenever the logical formula has unbound variables, we need a construct to come up with applicable values, like for example a constraint solver.
- an update. This is triggered after the transition is performed, updating some variables.
- a label. This is the observable message that is seen when the transition is performed.

Some extra details on labels: Labels are also symbolic in that they can have label-parameters. These are (volatile) variables that can be used to transfer data from the label to the STS.

Literature (available for download from homepages.cwi.nl/~jve/courses/12/testing2012/advanced):

Example 3: Data mining and visualization

Problem
Running a test case on an application can result in information on the states and path the test case covered in the application, the effects the test had, whether the test passed etc. Being able to see patterns in the wealth of information provided by tens of thousands of test cases can provide valuable insight in the weak spots and causes of errors of an application.
To be researched
Look into data mining and data visualization techniques for detecting and presenting patterns in large sets of test results. Evaluate the techniques using real life data. Select and apply data classification and clustering tools and libraries (for example RapidMiner, Weka) to analyze and structure test results. Select and apply visualization tools (for example Walrus, GUESS, Improvise) to present the (structured) test results in a meaningful way.

Evaluation
The project can use test results from a range of customer applications differing in complexity and with real bugs. Project results can be discussed and evaluated with domain experts that are able to judge the meaningfulness of patterns found. Axini's interaction designer can be consulted on visualization issues.

Context
With a large number of test cases it becomes unfeasible for testers to manually review each test case. Software should help by classifying and clustering similar test cases, by identifying probable causes of errors, other relevant relationships etc.

Example 4: Formal DSLs

Problem
The Ruby programming language allows easy implementation of DSLs, that can always fall back on the power of the full language to perform operations that cannot (easily) be captured by a DSL. Unfortunately, Model-based testing theory is usually based on state machines or transition systems and as such that theory does not apply to models written in Ruby DSL's. For expressing their models, DSLs are preferred by our customers. As rigorous testing requires the theory to be applicable to the models, a way to bring these two together is required.

Note that a lot of research and knowledge about DSLs is available at the UvA, more specifically at the CWI in the group of Professor Paul Klint.

To be researched
Is there a transformation or morphism that describes a correspondence between a Ruby DSL and a symbolic transition system (our poison of choice)? There are tools to construct the Abstract Syntax Tree of a piece of Ruby code, which resembles a series of state machines that operate on a set of overlapping data stores. Can this notion be formalized into a symbolic transition system?

Evaluation
Tools from Model-based testing theory can be used to guide test case generation by Ruby models. The amount of test coverage achieved can be expressed in terms of states and transitions.

Example 5: Modeling and testing with decision tables

Problem
Formal modeling of functions in a pragmatic and user friendly way is difficult in a transition system setting. Decision tables are an interesting formalism that seem to enable user friendly (as in also for non-technical people) modeling of functions in a formal way. They are Turing complete and in that sense should be even powerful enough to model entire systems. We are very interested in how we can apply decision tables for model-based testing (and programming in general).
To be researched
How can we use decision tables as a formalism for model-based testing? How do we derive test-cases for decision tables? Is system modeling with decision tables a viable approach or does it become too complex? Can we combine our existing transition system approach with decision tables, getting the benefit of both worlds?

Evaluation
After the literature study you will build one or more prototypes to test your approach. We have several transition system models to compare your approach against. As far as we know this topic has not been researched before, which makes it possible to make a scientific paper out of the results.

Example 6: Visualization-based modeling versus textual models

Problem
Axini currently has a textual model editor, because textual models are usually very easy to refactor and to share with others. However, they can be hard to understand for someone who isn't a programmer. Visual models are usually easier to understand, but usually harder to maintain.

To be researched
How can we have best of both worlds?
Determine how existing visualization-based modeling techniques, such as UML or POOSL can be used or extended for using them for model-based testing.

Evaluation
One of our clients is the leading manufacturer in self-scan checkouts. You will apply the developed technique to the systems of this client. You will compare the models and generated tests of the developed technique with the results from non-visual modeling.

Context
The challenge is to build models that on the one hand provide a (high level) system overview, and on the other hand capture all details needed for thorough testing and test derivation.

Example 7: Test coverage in practice

Problem
Testing software is inherently incomplete; no matter how intensely a product has been tested, there remains some probability that one or more faults are present. Still, observing correct behavior increases our confidence in the correctness of a system. To quantify this increase of confidence, several notions of 'test coverage' and 'risk' have been developed. However, many of these notions are very basic, and therefore do not predict the quality of a test process in a very accurate manner.
To be researched
Recently, a more complex notion of test coverage has been developed. It takes a probabilistic approach and is based on sound mathematical foundations. Unfortunately, this notion requires the tester to specify a large amount of information about the system-under-test, limiting its use in practice. Moreover, dependencies between faults are not yet taken into account, reducing its accuracy.

The purpose of this assignment is to extend and simplify the existing test framework, making it more practically applicable (by approximating some of its less relevant input parameters) and more accurate (by including fault dependencies). A practical case study will be performed in order to find useful approximations and often-occurring fault dependencies. This case study can be done in co-operation with the FMG group of the University of Twente, specialized in formal testing.

We worked on this topic before with a graduate student and got very interesting results. We look forward to develop more theory and tools in this area.

Example 8: Modeling investment banking with Alloy

Problem
We want to model software systems in a way that
1. results in models that can be understood by business users;
2. is suitable for modeling data-intensive applications;
3. allows us to automatically generate test cases from the model that have a high test coverage of the software system.

To be researched
Evaluate if Alloy (alloy.mit.edu) and its TestEra tool set can meet these requirements. Study test case generation with Alloy and TestEra. Prototype a test case generator that analyzes Alloy models and generates test cases. Apply and evaluate the prototype using a real life case from an investment bank.

Evaluation
The prototype will be used to model part of a real life system of one of Axini's customers, an investment bank. The modeling process and its test results will be compared to existing results of other approaches. Customer (business) users can provide feedback on the model(s).

Context
To reliably determine if a software system behaves as specified can require tens of thousands of test cases. Manually designing or executing such numbers is not feasible. Model-based testing techniques can automatically generate and execute large numbers of tests. Modeling real, data-intensive systems is still a challenge.

---

1 Timmer, Stoelinga; Interpreting a successful testing process: risk and actual coverage; May 2009.
General topics that interest us

In case you like the topic of model-driven software engineering, but you did not yet find the perfect topic, come and talk with us. Here are some topics, definitely not a complete list, that we are interested in:

- Implementation of domain specific modeling languages.
- Semantics of modeling languages.
- Case studies of Model-Based Testing applied to the systems of one of our customers.
- Implementation of a modeling language: decision table semantics, combined with symbolic state machines.
- Model-checking.
- Reachability in symbolic transition systems.
- Optimization of symbolic transition systems.
- Implementation of a modeling language with time (for example with timed automata).
- Software engineering.
- Agile software development.