The Role of Computer Science and Software Technology in Organizing Universities for Industry 4.0 and Beyond

Prof. dr. ir. Mehmet Aksit
m.aksit@utwente.nl
Department of Computer Science, University of Twente,
Enschede, The Netherlands,
http://www.utwente.nl/ewi/trese/people/Aksit/
http://fmt.cs.utwente.nl/

Contents

- Making strategic plans of companies reality!

Approach:

- The company maturation process;

Justifications:

- Motivation 1: Researchers perspective;
- Motivation 2: Trends in industry
- Motivation 3: Re-shaping universities
- Motivation 4: CS
- Motivation 5: Software engineering and technology

- Conclusions.

The beginning of all!

The problem statement

Higher-level objectives

Given a set-of (potentially) strategically important companies

- How to determine the strategic objectives of these companies?
- How to turn the strategic objectives into tactical and implementation objectives?
- How to steer the overall process and be sure that it is converging to these objectives?

Possible answers

- How to determine the strategic objectives of these companies?

I think companies should know better than me, but they must be aware of the trends! you know software is crucial:
every company will be a software company in the future;

- How to turn the strategic objectives into tactical and implementation objectives?

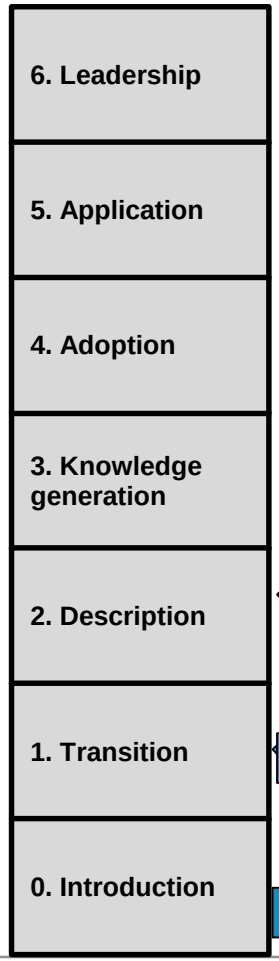
Apply problem solving processes;

- How to steer the overall process and be sure that it is converging to these objectives?

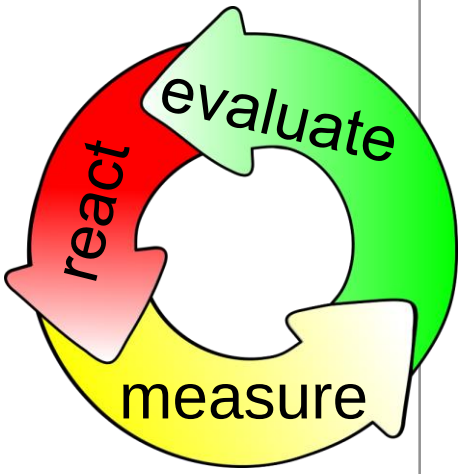
Define a process that: (1) classifies companies according to their maturity levels and identify the current level; (2) define maturation processes; (3) steer the overall process!

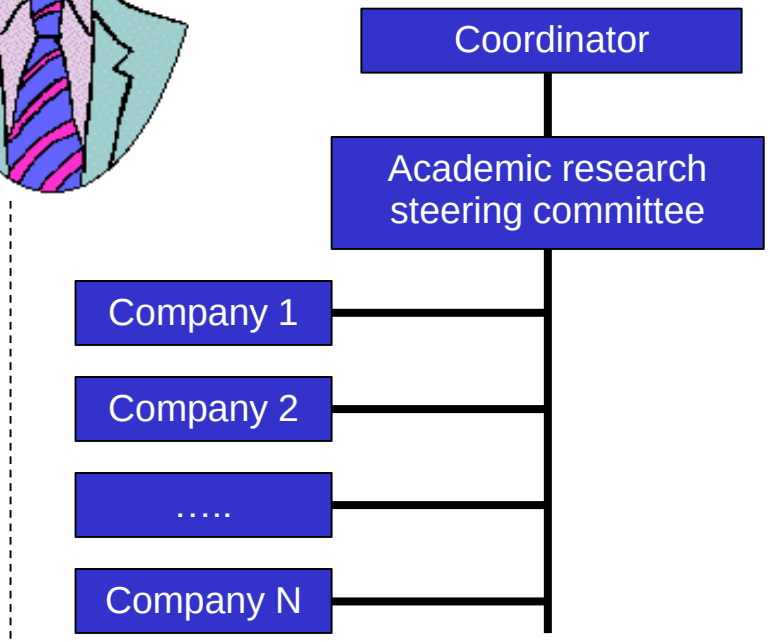
The maturation process

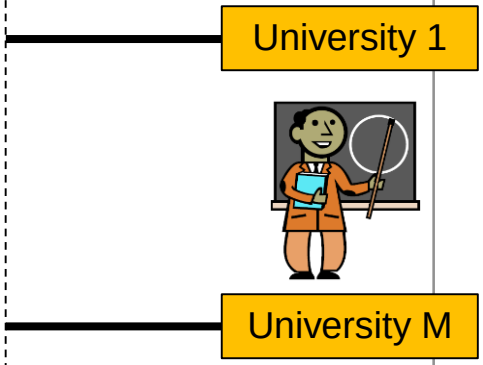
The intention is to enhance the maturity of software techniques by guiding the companies along this process



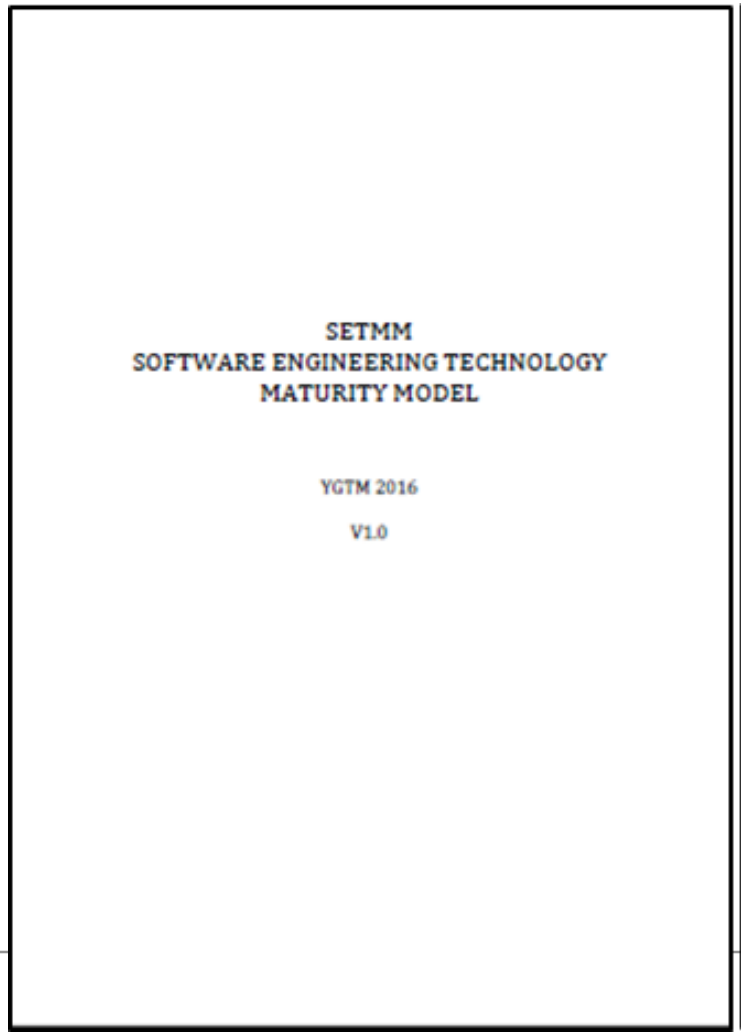
Looks like a feedback control system







The process is described and approved as a report



SETMM
SOFTWARE ENGINEERING TECHNOLOGY
MATURITY MODEL

YGTM 2016

V1.0

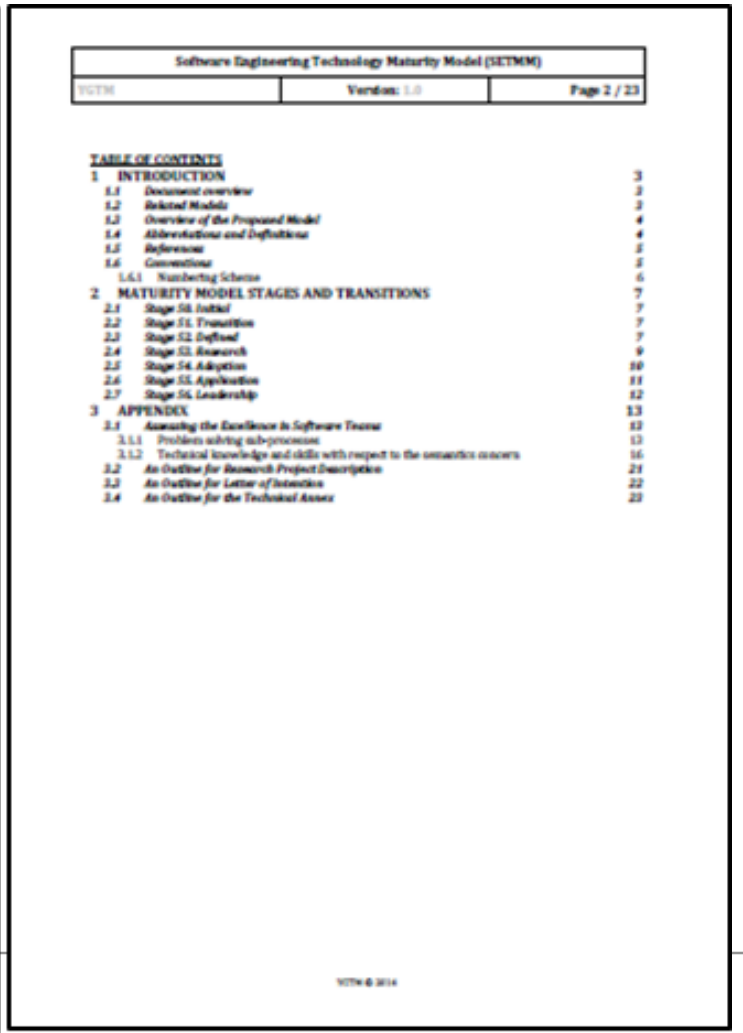


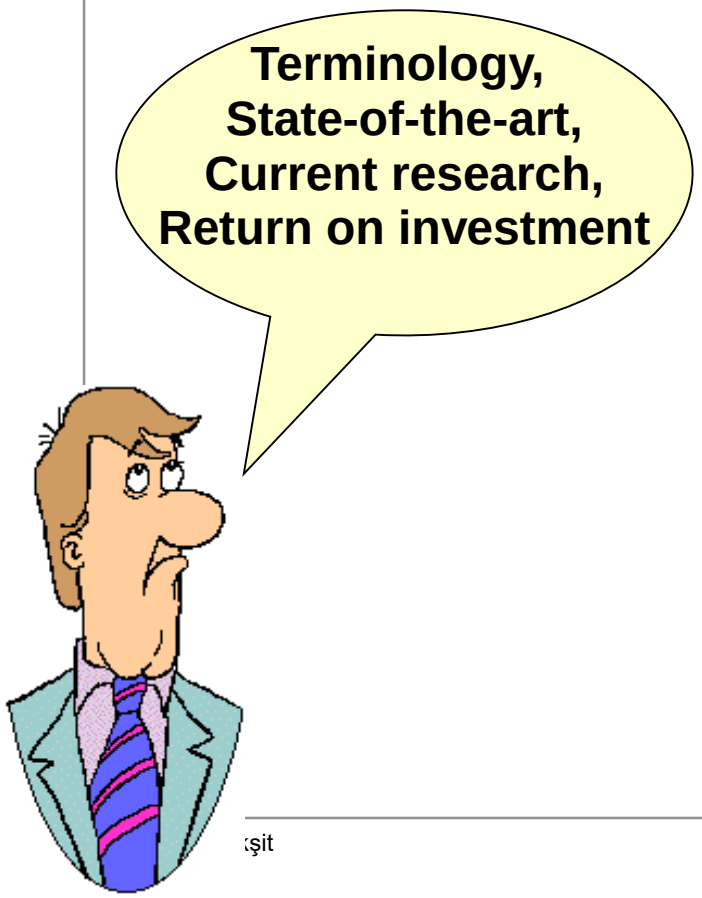
TABLE OF CONTENTS

(1) companies present their strategic plans to a team of researchers

The intention is to identify the obstacles/challenges in reaching the objectives!

(2) Researchers and company engineers work together extensively to identify the (future) challenges and prospective solutions

(3) Reports are written, discussed, modified and accepted, by describing the definitions of the terminology, the context, the observed research challenges and the related research work



A sample document

Mapping concerns from the business domain to CS research domain!

The Project "Model-Based Testing, Runtime Verification and Debugging for Verifying Product-Lines" (TRDP)

Version: 1.0

Keywords: model-based verification of software systems, verification of product-lines, reducing false positives and false negatives in verification, reducing efforts in defining models, scalability versus precision, verification of software in execution environment, model-based testing,runtime verification, debugging, multi-model multi-technique verification techniques.

Category: *Model-based verification*. The projects in this category are:

- MCAP (Integration of Model-Checking with Complementary Approaches for Verifying Product-Lines; for On-Board Systems)
- TRDP (Model-Based Testing, Runtime Verification and Debugging for Verifying Product-Lines; for On-Board Systems)
- MVVP (Methods for Model Based Verification and Validation Techniques for Product-Lines; for On-Board Systems)
- ASVC (Agent System Verification and Control; for Simulation Systems)
- AEVF (Adaptive Enterprise Verification Framework; for Mission-Critical Enterprise Systems)
- SCER (System-Wide Constraint Enforcement and Resolution; for Systems of Systems)

There is a relation with:

- SPLA (System product-line engineering; for On-board systems): Verification challenges of product-lines are considered;
- ASVC (Agent System Verification and Control; for Simulation Systems): Although specification languages will be tailored to the needs of agent-based simulation systems, the underlying mechanisms will be partly based on runtime verification techniques.
- AEVF (Adaptive Enterprise Verification Framework; for Mission-Critical Enterprise Systems): Although specification languages will be tailored to the needs of mission-critical enterprise systems, the underlying mechanisms will be partly based on runtime verification techniques.
- AVWF (Adaptable and Verifiable Workflow Framework; for Mission-Critical Enterprise Systems): Workflow specifications may be (partly) verified using runtime verification techniques.

errorsearly, some detected errors may actually not happen in practice (false positives) due to unspecified environmental conditions, practical constraints, etc. In the model. Also, certain errors may remain undetected (false negatives) if the model does not include the relevant portion of the software necessary to detect the error. Moreover, model verification can be impractical due to the complexity of the model. Therefore, verification of software in execution environment is considered a necessity.

We consider model-based testing, runtime verification, and debugging as three important approaches in verifying software in execution environment.

In model-based testing [1], models are used to either represent the desired behaviour of a system under test, or to represent testing strategies and a test environment. Various different kinds of modelling techniques can be used such as state-machines, constraint logic programming, symbolic execution, model-checking etc. The selection of the model can depend on the specific characteristics of software, its testing environment and the desired testing method to be used. One may adopt various ways to apply model-based testing in practice, such as online testing, offline generation of executable tests, and offline generation of manually deployable tests. Model-based testing aim at offering a more systematic and semantic approach in covering the critical parts of software under test than traditional dynamic testing.

In runtime verification [2], information is extracted from a running system and verified against the desired properties of the system. In this approach, verification of software is generally carried out in its actual deployment and execution environment. Various models can be used to specify the desired and undesired properties of software. It is a common practice to automatically generate the appropriate runtime verification environment from the specifications.

Debugging is defined in [3] as "The issue of debugging is to identify the infection chain, to find its defect, and to remove the defect such that the failure no longer occurs". A defect, which is also called the root cause (error), is a mistake made by programmersin the code that can causeinfections. Infections may occur after the defect is executed. An infection is a deviation of theactual execution from the programmers' expectation. It propagates in the following execution and leads to other infections. A failure is an observable unexpected symptom, such as a wrongbehaviour or a wrong program state. The cause-effect chain from defect to failure is called aninfection chain. There are various debugging techniques introduced in the past such asinteractive debugging, trace-based debugging, static slicing, etc. [4, 5].

In addition, well-definedsyntactic and semantic interfaces must be defined to the model-checking tools which will be developed within the context of the MCAP project. It is therefore important to develop methods, tools and techniques to effectively identify the strong and weak points of each technique so that the verification processes can be shared among them. The challenge in a multi-model/multi-technique verification approach is to minimize the number of models to be defined, since modelling is a costly activity. For this purpose, model transformation techniques can be used, where appropriate. Finally, defining models must be intuitive and usable for the software engineers.

Return on Investment

For companies that are developing safety-critical and mission-critical systems, correct functioning of software systems is at most important. Unfortunately, by using traditional programming methods and techniques, it is impossible to guarantee that complex systems have zero non-critical faults. To test such systems dynamically is very costly and requires long testing periods. Despite exhaustive testing efforts, due to almost unlimited combinations of execution paths and data value variations, certain kinds of critical errors may still remain undetected. Moreover, mostmodern systems are open real-world systems in the sense that they are subject to integration with other systems and they have to deal with continual evolution demands. Furthermore, verification of product-lines with many possible product variations can make the verification process for each product very costly.

This project aims at enhancing verification techniques in the execution environment of software systems; it will provide an integrated and uniform knowledge and experience in the advanced model-based testing, runtime verification and debugging techniques combined together, so that critical errors can be detected in the execution environment. Moreover, these techniques will be integrated with model-based verification techniques such as model-checking. To be able to realize an effective technology transfer to the current projects, this projectwill be equipped with the following measures:

The research activities aim at improving the skills instead of searching for a dedicated solution for a given project. This helps in adopting the obtained knowledge in different projects as well.

The activities will deliver various models of the example systems considered, techniques described in the deliverable reports and proof-of-concept implementations. In addition, a workbench will be developed that integrate various tools. These tools will be based on open source tools, where possible.

The project will adopt real examples from the company. This will increase the relevance of the research project.

- SCER (System-Wide Constraint Enforcement and Resolution; for Systems of Systems): Although specification languages will be tailored to the needs of system of systems applications, the underlying mechanisms will be partly based on runtime verification techniques.

Application areas: All, but particularly safety-critical and mission-critical systems.

Application focus: On-board systems.

Number of Ph.D.'s planned: 3

Part 1: Description of the Project

Motivation

With the help of traditional programming practices, it is almost impossible to write correct programs at least for the following four reasons. First, current software systems are large and complex. This makes it very difficult for the software engineer to comprehend software in detail and as a whole. Second, modern software systems are usually distributed over multiple processors/nodes and execute concurrently. Concurrently executing software systems have to be frequently synchronized with each other. This may create time-dependent errors which are in general very hard to detect. Third, software systems are not designed for a single release and as such they are subject to continuous evolution. New requirements, possible improvements to software and bug fixes increase the complexity of software further. Last but not least, product-line approaches are becoming more and more common. Software is generally developed not as a single product but as a family of products. Due to the possibility of a large variety of configurations of products, assuring correctness of each individual product may be very costly to realize.

Today's common practice is to detect software errors through dynamic testing of software. By dynamic testing alone, it is extremely difficult to capture the errors introduced during the programming phase, since modern software systems contain practically infinite number of execution paths.

Description

There are two basic concerns in verifying software systems: to reduce false positives and false negatives. A false positive means assuming an error while there is in fact no error; a false negative means missing an error while in fact there is an error.

We classify verification techniques into two complementary categories: verification based on abstract models of software and verification based on (part-of) actual software in execution environment. Although verification on abstract models can be useful in detecting software

Since model-based testing, runtime verification, and debugging techniques are largely complementary, the project TRDP aims at combining thesetechniques to reduce false positives and false negatives in the verification of software in its execution environment. However, the combination of these techniques should not result in an unacceptable overhead due to the definition of multiple models and due to setting up different verification environments.

Research challenges

Traditionally, the verification techniques such as model checking, static analysis, model-based testing, runtime verification, and debugging are studied by separate communities and as such methods and tools developed for these techniques are not well integrated with each other. Moreover, verification of open andevolvable product-line systems have not yet received the desired attention, although verification of product-lines applied for safety-critical and mission-critical systems is considered essential.

Each of the verification techniques has its own challenges. For example, in model-based testing techniques, based on the test requirements, selecting the right modelling technique at the desired level of abstraction, and mapping the abstract test cases to the concrete ones are not trivial. Also, defining the required models can be too labour intensive. Moreover, there may be specific limitations of the selected modelling technique. For example, if model-checking techniques are selected, the verification process may not be scalable due to the complexity of the model [6].

In runtime verification techniques, research activities have been focused on creating usable verification environments, generating efficient and effective monitors and verification mechanisms, defining expressive specification languages, and coping with distributed environments and multi-language environments [2, 7-12].

In debugging, defining expressive languages for breakpoints, dealing with domain specific, dynamic and/or distributed languages, and defining semantic models and techniques to trace the root cause of failures have been considered as important challenges [3, 4, 13-15]. Research activities on automated diagnosis techniques can be utilized for this purpose [16, 17].

This project aims at addressing these challenges within the context of product-lines.

In addition to the above mentioned research challenges, the TRDP project has its specific challenges to overcome. The most important challenge is to seamlessly integrate the model-based testing, runtime verification, and debugging techniques together, so that each technique complements the others in the verification of software in its execution environment.

The project will carry out a number of experiments using realistic cases. The results of the pilot projects will be considered to improve the research results.

There will be a set of continuous technology transfer activities by the people who are involved in the research activity by means of participating to lectures, courses and involvement in the ongoing projects.

References

[1] Model-based testing, Wikipedia http://en.wikipedia.org/wiki/Model-based_testing

[2] Run-time verification, Wikipedia http://en.wikipedia.org/wiki/Runtime_verification

[3] Zeller, A. (2005) Why Programs Fail: A Guide to Systematic Debugging. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2, 127

[4] Yin, H. (2013) Defusing the Debugging Scandal - Dedicated Debugging Technologies for Advanced Dispatching Languages, Ph.D. Thesis, University of Twente.

[5] Silva, J. (2011) A Survey on Algorithmic Debugging Strategies, Advances in Software, Volume 2, Issue 11, pp. 976 - 991

[6] Tretmans, J. (2007), Tangram: Model-based Integration and Testing of Complex High-Tech Systems, Embedded Systems Institute, 2007

[7] Somayeh Malakuti and Mehmet Aksit, "Event Modules: Modularizing Domain-Specific Crosscutting R.V. Concerns", ACM Transactions on Aspect-Oriented Software Development, Special Issue on Runtime Verification and Analysis, 2013 (to appear)

[8] Malakuti Khah Olun Abadi, S. and Akşit, M. and Bockisch, C.M. (2011) Distribution-Transparency in Runtime Verification. In: Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops, ISPAW 2011, 26-28 May 2011, Busan, Korea. pp. 326-335. IEEE Communications Society.

[9] Malakuti Khah Olun Abadi, S. and Akşit, M. and Bockisch, C.M. (2011) Runtime Verification in Distributed Computing. Journal of Convergence, 2 (1). pp. 1-10.

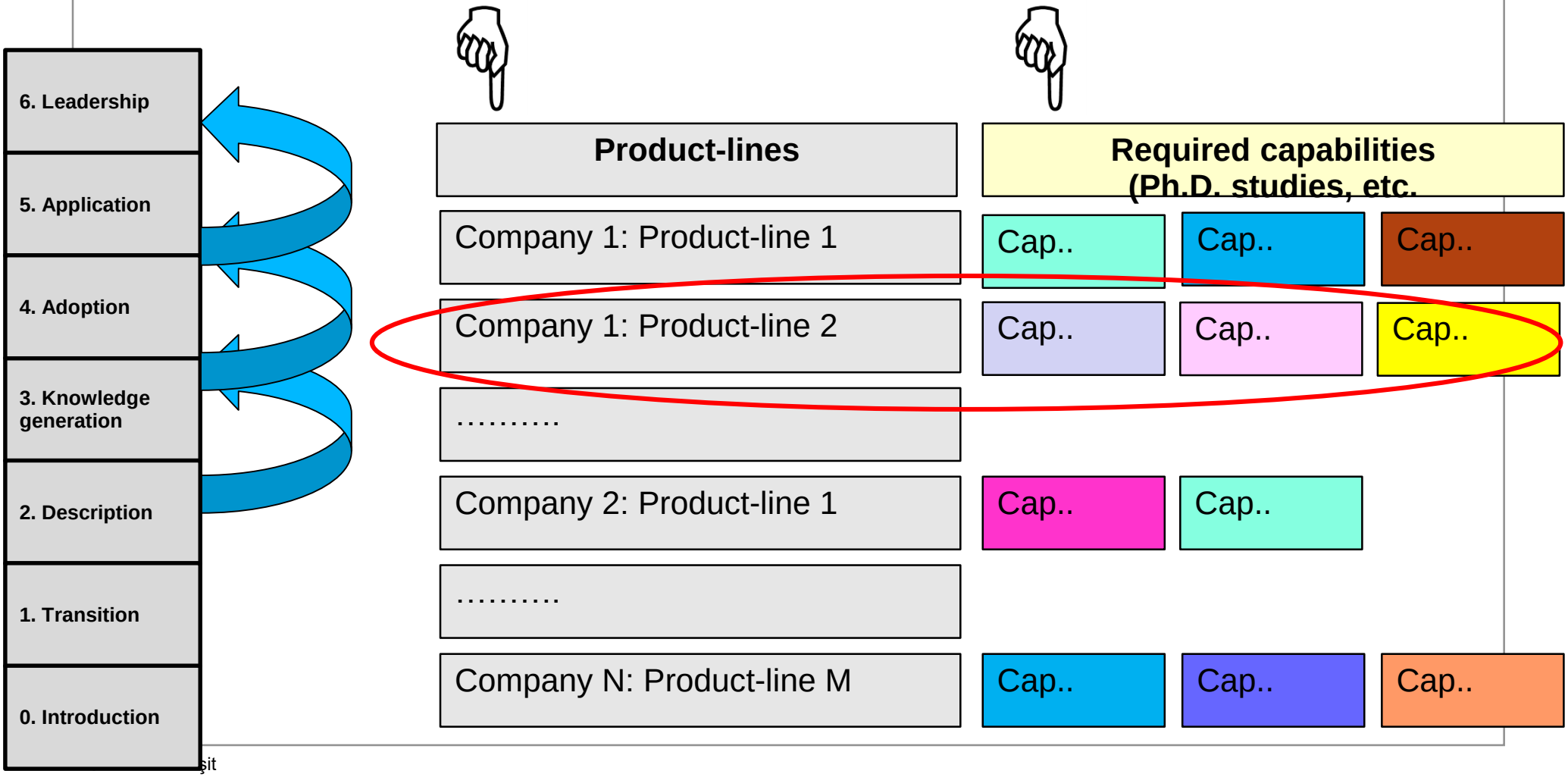
[10] de Roo, A.J. and Sözer, H. and Akşit, M. (2011) Runtime Verification of Domain-Specific Models of Physical Characteristics in Control Software. In: Fifth IEEE International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 27-29 Jun 2011, Jeju Island, Korea. pp. 41-50. IEEE Computer Society.

[11] Sözer, H. and Hofmann, C. and Tekinerdogan, B. and Akşit, M. (2011) Runtime Verification of Component-Based Embedded Software. In: Proceedings of the 26th International Symposium on Computer and Information Sciences, 26-28 September 2011, London, UK. pp. 471-477. Springer Verlag.

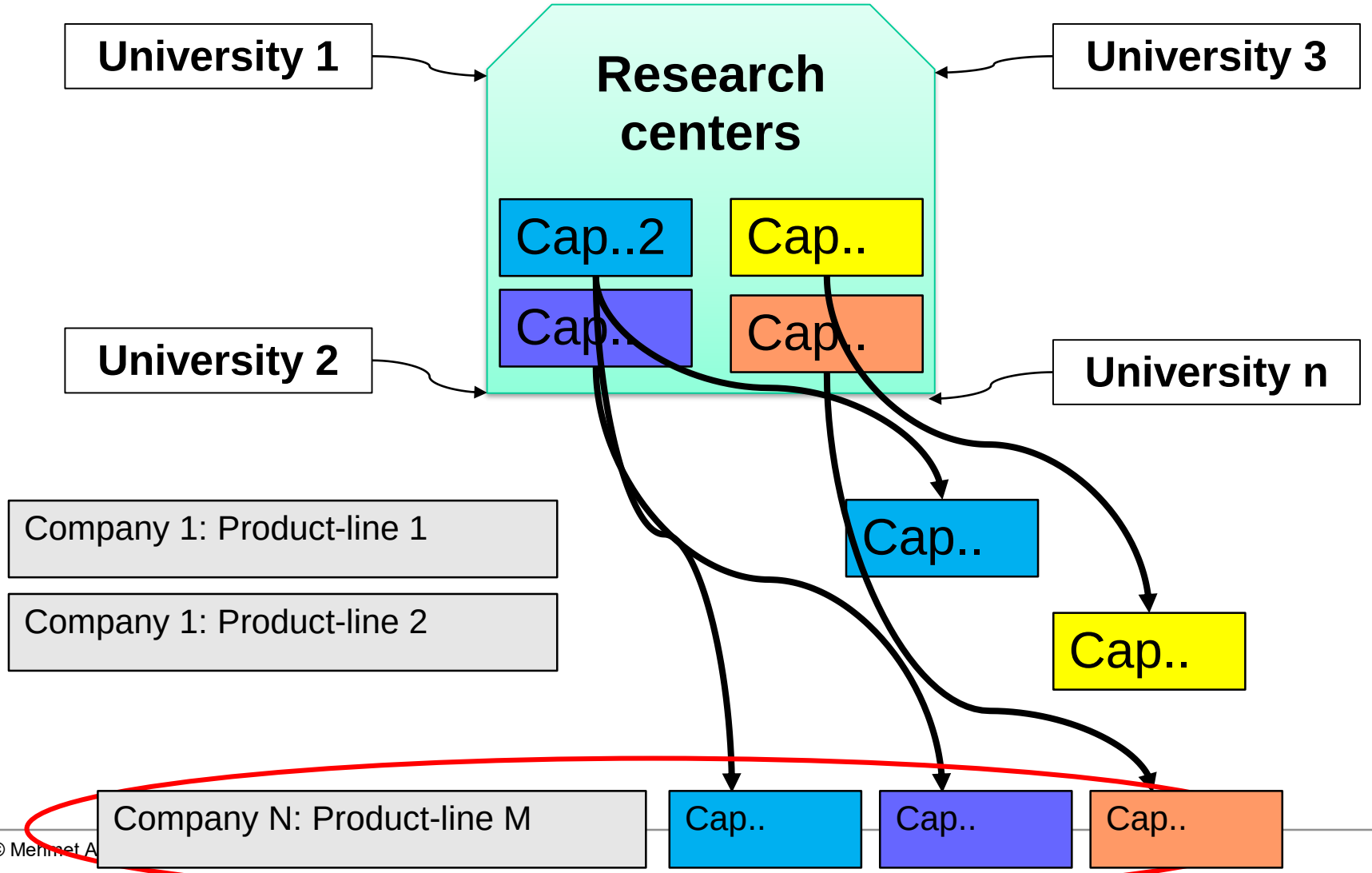
[12] Malakuti Khah Olun Abadi, S. and Bockisch, C.M. and Akşit, M. (2009) Applying the Composition Filter Model for Runtime Verification of Multiple-Language Software. In: The 20th annual International Symposium on Software Reliability Engineering, ISSRE 2009, 16-19 Nov 2009, Mysore, India. pp. 31-40. IEEE Computer Society.

[13] Yin, Haihan and Bockisch, C.M. and Akşit, M. (2013) A Pointcut Language for Setting Advanced Breakpoints. In: Proceedings of the 12th Annual International

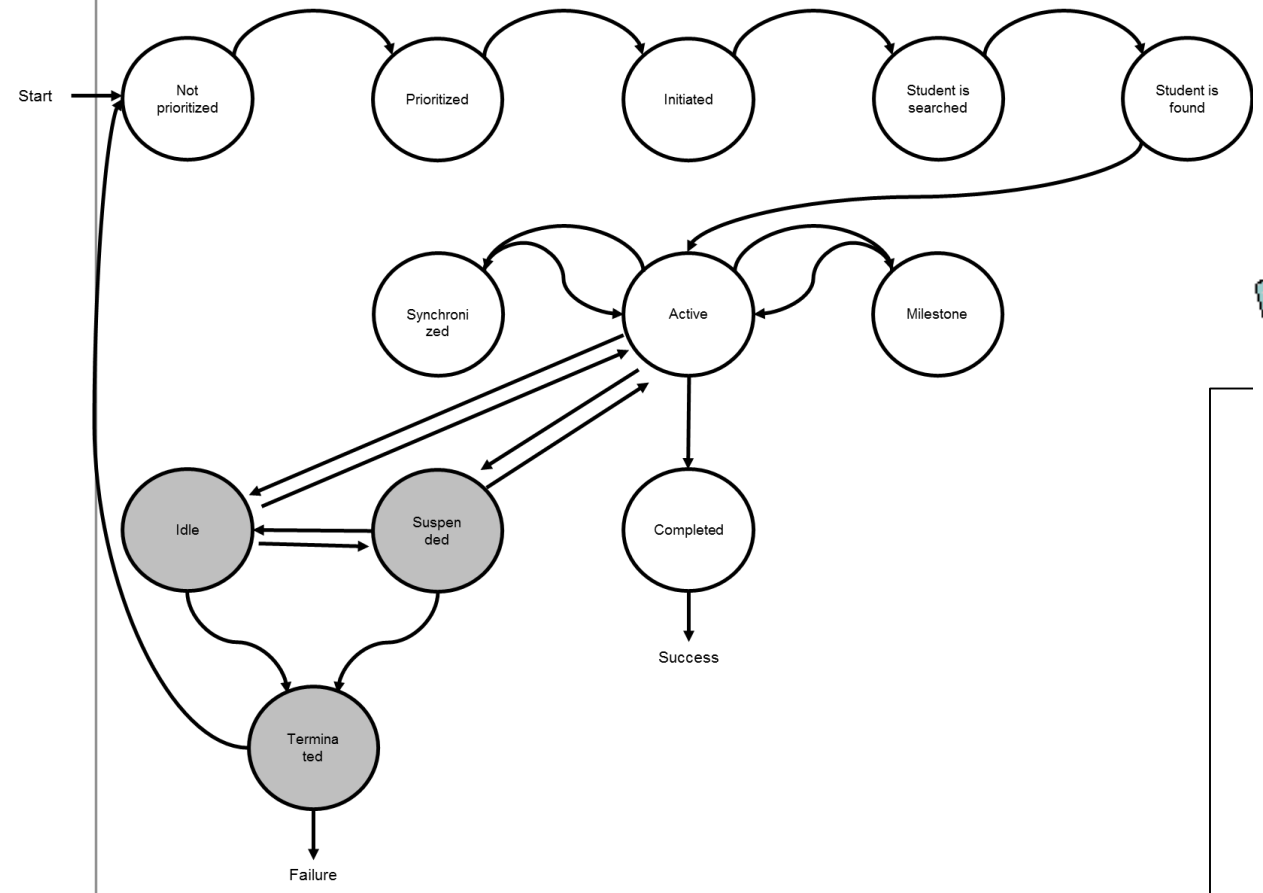
Research activities are grouped around the product-lines



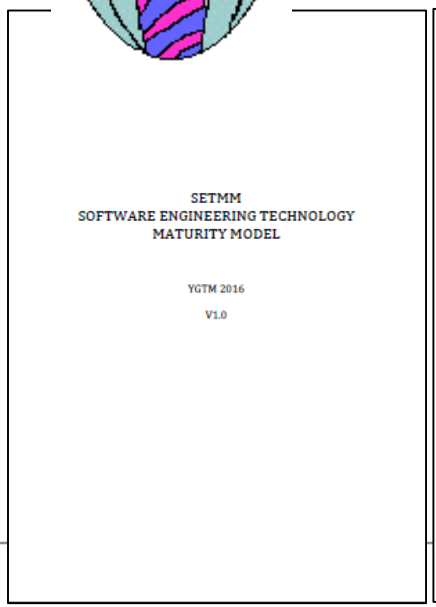
Company-academia coordination (research) centers



A reactive feedback-control process model is defined; An example: part of the process for level 3



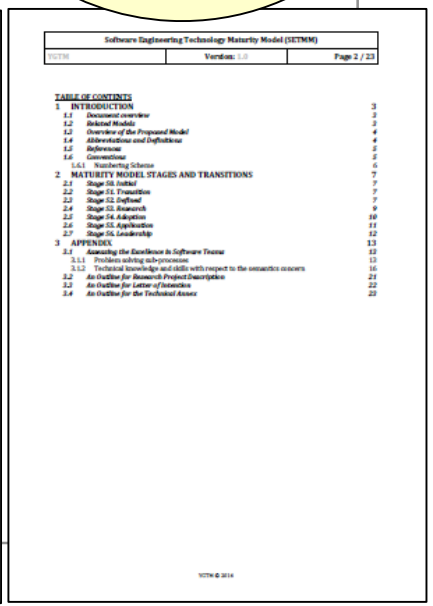
The committee is also responsible for synergy & coordination & scientific progress



SETMM
SOFTWARE ENGINEERING TECHNOLOGY MATURITY MODEL

YGTM 2016

V1.0



Software Engineering Technology Maturity Model (SETMM)

YGTM | Version: 1.0 | Page 2 / 23

TABLE OF CONTENTS

The technical results

The Product Domain Categories

Company 1:

- On-board systems (14 Ph.D.'s);
- Simulation systems (10 Ph.D.'s);
- Mission critical enterprise systems (14 Ph.D.'s);
- Mission critical systems of systems (10 Ph.D.'s);

Company 2:

- Avionics systems (15 Ph.D.'s);
- Communication systems of systems (9 Ph.D.'s);
- Mission-critical hybrid control systems (6 Ph.D.'s);
- Sensor fusion systems (3 Ph.D.'s);

Company 3:

- Intelligent platforms (8 Ph.D.'s);
- Distributed simulation (6 Ph.D.'s);
- Intelligent control systems (6 Ph.D.'s);

Company 4:

- Modernization of flight systems (11 Ph.D.'s);
- Satellite systems (14 Ph.D.'s);
- Unmanned aerial systems (17 Ph.D.'s);
- Generic fault tolerant computing (2 Ph.D.'s);
- Generic fault-removal techniques (5 Ph.D.'s);

Including myself, this work has been carried out together with:

Prof. dr. ir. Bedir Tekinerdoğan;
Assoc. prof. Hasan Sözer;
Hakan Faruk Safi;
Meryem Ayas.

The Research Categories

- Adaptability and reuse (25%)
- Model-based verification (24%)
- Domain-specific architectures (12%)
- Domain-specific languages (10%)
- Design methods (8%)
- System design (6%)
- Model building (6%)
- Data/cloud management (3%), Application frameworks (3%), Architecture business assessment (3%):

Motivation 1:

from

Researcher's point of view

Needs for advanced research in companies

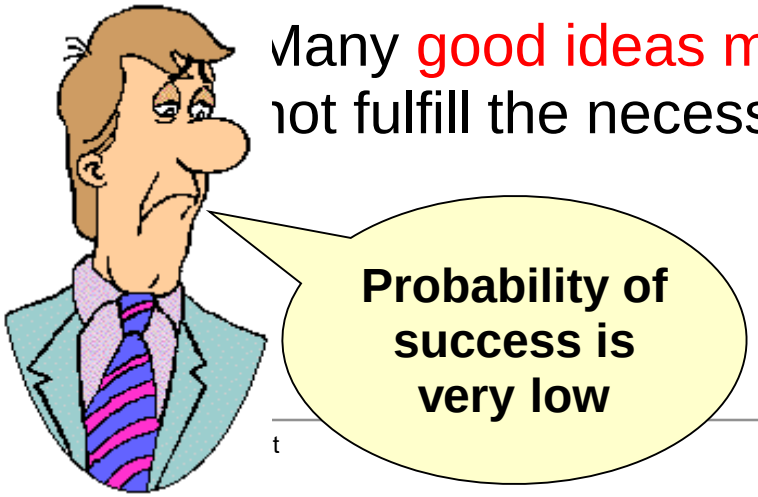
- Computer science and technology advances rapidly.This makes software developing companies difficult to follow the advancements effectively.
- Tight project schedules and limited budget for research and development make this process even harder.
- As a result, companies have difficulties in achieving the excellence in computer science and software technology to overcome their technical challenges.

Methods used in research granting programs

Current university-industry collaborations are mostly based on governmental grants of university research programs, Joint Ventures (JVs) and Framework Programs (FP6, FP7, etc.).

- The existing university-industry cooperation methods are based on writing grant proposals to financing organizations and passing through strict selection processes. This is, in general, a very inefficient and tedious process to undertake.

Many good ideas may not go through simply because they do not fulfill the necessary procedural requirements.



From the researcher's perspective

- Researchers do not waste time in writing long and tedious grant proposals;
- Researchers are directly involved in the maturation process of the companies;
- Scientific quality is monitored and guaranteed by the steering committee, but

  - the universities & researchers must agree on the way of working!

Motivation 2:

from the perspectives of the

Trends in industry

Trends (i)

1. Knowledge intensive manufacturing **instead of** labor/resource-intensive manufacturing.

2. Focus on owning and managing knowledge and skills and intellectual property rights **instead of** focusing on labor/resource intensive manufacturing processes.

3. Dynamically managed and optimized, multi-asset portfolio **instead of** fixed/ad-hoc, single-asset portfolio.

4. Mass customization **instead of** mass production.

Trends (ii)

5. Proactive self-organizing companies **instead of** inflexible hierarchically-organized companies.

6. End-to-end alignment and optimization of (manufacturing) processes **instead of** focusing only on the improvement of individual phases.

7. Multi-disciplinary usage of teamed personnel **instead of** working with solely operating individuals.

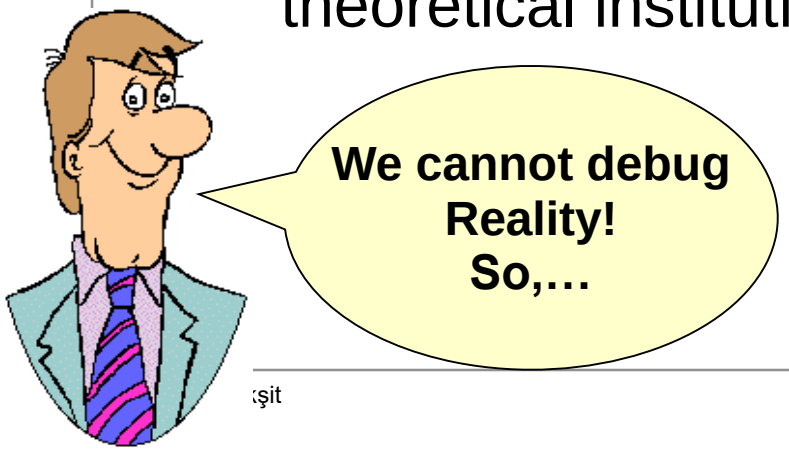
8. Organizing businesses/enterprises globally through networks **instead of** isolated and/or localized organizations.

Trends (iii)

9. Improved time-to-market **instead of** long sequences of research, design, manufacturing and marketing phases.

10. Intensive use of state-of-art CS-ST as the “main enabler” of modern businesses **instead of** considering CS-ST just like any other technical skill.

11. Strong cooperation with universities for the purpose of innovation **instead of** considering universities mainly as theoretical institutions that educate people.



What about Industry 4.0?

- Interoperability meaning that sensors, devices, machines, and people can connect and exchange information with each other.
- Information transparency meaning that a rich set of data can be gathered from various sources.
- Technical assistance meaning that machines, systems, processes, human beings, etc. can be intelligently and effectively assisted to monitor, control and optimize the overall manufacturing process.
- Decentralized decisions meaning that subsystems can autonomously take decisions where possible.

What they said about Industry 4.0

- With Industry 4.0, it is expected that machines and systems will become more self-aware and self-learning so that their effectiveness and maintenance can be improved.
- Due to networked data gathering and intelligent and autonomous process control, the manufacturing processes will be much more efficient and effective than traditional manufacturing processes.
- It is claimed that Industry 4.0 is the 4^{th} industrial revolution in the history of manufacturing.

Our assessment of Industry 4.0

We think that the concepts relevant to Industry 4.0 must be defined and understood in the process of on-going transition from traditional to modern manufacturing processes. It is important to stress that such transitions are not abrupt in nature but gradual, depending on the characteristics of manufacturing, technological and societal progresses.

- It is clear from “the list of trends” in this presentation that Industry 4.0 refers to (or a new name of) a part of a natural transition in manufacturing processes which has been taking place since several decades. We therefore term “the list of trends” as “a list of attributes of manufacturing processes for Industry 4.0 and beyond”.

Motivation 3:

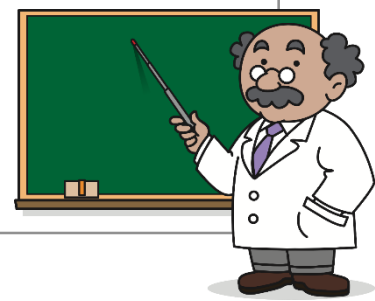
for

Re-shaping universities for the future

Traditional universities

- Research, where academic personnel of the university are expected to be expert in certain fields. The selection of the topic of a field is not necessarily derived from industrial and societal needs; it can be ad hoc. The expertise is quite specific and theoretical. The excellence is measured according to number of publications in certain pre-classified journals.

- Education, where academic personnel of the university are expected to give lectures in their fields of expertise and examine the students by oral and written tests. In addition, students are expected to be supervised in writing their theses.



How universities should be! (i)

1. The specializations must be derived from the needs of the targeted society and industry instead of ad-hoc selection of topics;

2. The academic personnel must learn to work together in multi-disciplinary teams.

3. The academic personnel must be proactive in forming networks to cooperate with national and international institutions and colleagues not only from his/her own discipline but also from other disciplines.

4. The academic personnel must be flexible enough to adapt themselves in changing demands from industry and society.

How universities should be! (ii)

5. The education process must be tailored to answer the mid-term and long-term needs of industry and society:

   - It must focus on the core concepts **instead of** hypes.
   - It must focus on gaining analytical skills, critical thinking and reasoning **instead of** memorizing what are in the books.
   - It must aim at teaching problem solving/synthesis **instead of** gaining knowledge which cannot be utilized for solving actual problems.
   - It must emphasize working in multi-disciplinary projects **instead of** only focusing on mono-disciplinary exercises.
   - It must enhance communication skills, such as oral and written presentation and argumentation skills **instead of** educating students with non-communicative and introvert attitude.
   - It must aim at increasing consciousness of students in ethical concerns **instead of** educating students with irresponsible and/or indifferent attitude.

How universities should be! (iii)

6. The university must create suitable organizational structures to enable the academic personnel efficiently and effectively fulfil the objectives listed above. These include:

- Proactive and self-adaptive organization to support the objectives of the university in dynamically changing contexts.
- Organization to set-up and carry-out multi-disciplinary projects for industry and society.
- Organization with an award system to motivate the academic personnel and students along the objectives of the university.
- Organization which emphasizes CS-ST since it is the “main enabler” of all disciplines at the university.

University industry, hand in hand..

Important quality attributes (i)

- Relevancy: The university must be highly relevant in addressing technical and social needs.
- Alignment with the current state-of-the-art research: The research and education activities to be carried out must advance the state-of-the-art so that the companies and businesses can be matured to be the leaders in their context.
- Cross-fertilization: Different university research and education activities can benefit from each other.

Important quality attributes (ii)

- Industry-as-laboratory: To identify the relevant problems and to test the proposed solutions, it is important that the principle investigators and the affiliated (Ph.D. and/or M.Sc., etc.) students visit the companies regularly and carry out experiments within industrial and societal context.

- Academic research steering committee: To coordinate the activities effectively and efficiently, it is important to monitor the progress of research and education activities and evaluate them with respect to the desired objectives.

Motivation 4

for

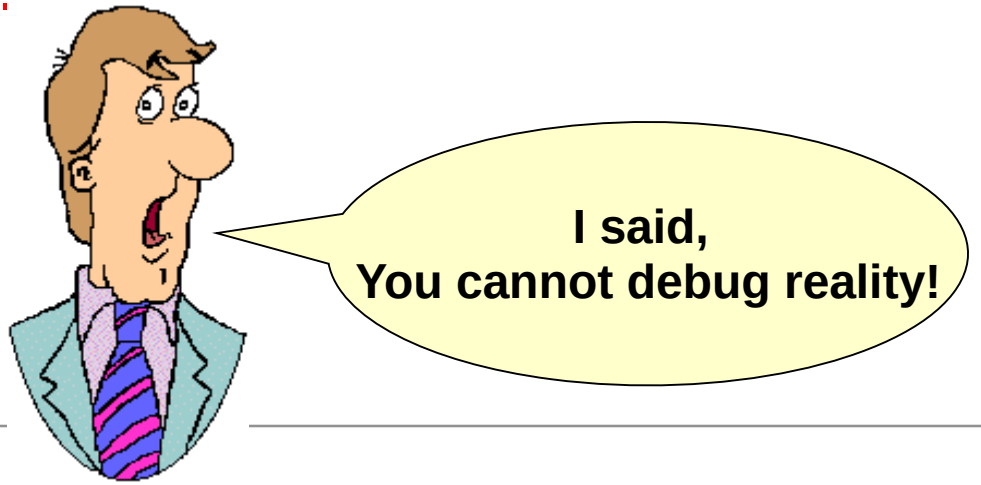
Computer Science!

Cannot be without CS

CS-ST is the main force in almost all industries; it creates added value for products and businesses.

There is almost no product in the market which does not contain software or is not produced by a process controlled by software.

To accomplish the objectives of Industry 4.0 and beyond, advanced CS-ST is needed.



Recent developments in CS

1. Large infrastructures, service-oriented architectures, cloud computing, systems-of-systems.
2. Sensors, Internet of Things (IOT), and pervasive computing.
3. Big data and big data analytics.
4. Security and cybersecurity.
5. Cyber-physical systems.
6. Artificial intelligence and related topics including computational intelligence, machine learning and multi-agent systems.
7. Graphical processing, and visualization including virtual reality.
8. High performance, and/or multi-core/parallel architectures including parallel programming.
9. **Theoretical and practical work on algorithms and/or constraint-based "solvers" to address a large category of mathematical problems. In general algorithms/solvers are applied to every category of computer science specializations listed in this section.**
10. **Software (engineering) methods and techniques to fulfil the functional and qualitative requirements of software systems. The concepts of software engineering can be applied to every computer science specialization listed in this section.**

Motivation 5

from the perspectives of

The role of software engineering and technology

The importance of software engineering and technology comes from the economics..

- Economical, sustainable and robust software systems which fulfil functional and qualitative requirements are essential for all software systems.
- To accomplish the requirements of Industry 4.0 and beyond, software engineering methods and techniques are crucial.
- No matter how intelligent a software solution is, if it cannot be realized with the desired quality attributes, one cannot expect an economical value out of it.
- As such software engineering methods and techniques can be defined as crosscutting (meta-level) concerns that relate to all developments within CS-ST.

The trends in software engineering and technology

Product-lines

Product-line **instead of** product design. Most products are developed and manufactured by specialized companies, which market families of products. It is not economical to develop each product from scratch.

Designing Reusable and Run-Time Evolvable Scheduling Software

Güner Orhan · Mehmet Akşit · Arend Rensink

Abstract Scheduling processes have been applied to a large category of application areas such as processor scheduling in operating systems, assembly line balancing in factories, vehicle routing and scheduling in logistics and timetabling in public transportation, etc. In general, scheduling problems are not trivial to solve due to complex constraints. In this paper, we consider *reusability* and *run-time evolvability* as two important quality attributes to develop cost-effective software systems with schedulers. Although many proposals have been presented to enhance these quality attributes in general-purpose software development practices, there has been hardly any publication within the context of designing scheduling systems. This paper presents an application framework called First Scheduling Framework (**FSF**) to design and implement schedulers with a high-degree of reusability and run-time evolvability. The utility of the framework is demonstrated with a set of canonical examples and evolution scenarios. The framework is fully implemented and tested.

1 Introduction

Scheduling is a decision-making process in which the *resources* are allocated to the *activities*. Scheduling processes have been applied to a large category of application areas such as processor scheduling in operating systems [43], car scheduling in elevator systems [35], facility scheduling at airports [40], antenna scheduling in radar systems [23], work-force scheduling in project management [30] and assembly line

Güner Orhan
University of Twente
E-mail: g.orhan@utwente.nl

Mehmet Akşit
University of Twente
E-mail: m.aksit@utwente.nl

Arend Rensink
University of Twente
E-mail: arend.rensink@utwente.nl

Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), Vienna, Austria, August 28–31, 2018

Systems of systems

Systems of systems **instead of** systems perspective. Software systems for Industry 4.0 are generally adopted in large distributed settings. Scale-ability and interoperability of systems are essential. Systems of systems architectures, are therefore the natural candidates of the platforms of Industry 4.0 architectures

Company 1:

- On-board systems (14 Ph.D.'s);
- Simulation systems (10 Ph.D.'s);
- Mission critical enterprise systems (14 Ph.D.'s);
- Mission critical systems of systems (10 Ph.D.'s);

Company 2:

- Avionics systems (15 Ph.D.'s);
- Communication systems of systems (9 Ph.D.'s);
- Mission-critical hybrid control systems (6 Ph.D.'s);
- Sensor fusion systems (3 Ph.D.'s);

Company 3:

- Intelligent platforms (8 Ph.D.'s);
- Distributed simulation (6 Ph.D.'s);
- Intelligent control systems (6 Ph.D.'s);

Company 4:

- Modernization of flight systems (11 Ph.D.'s);
- Satellite systems (14 Ph.D.'s);
- Unmanned aerial systems (17 Ph.D.'s);
- Generic fault tolerant computing (2 Ph.D.'s);
- Generic fault-removal techniques (5 Ph.D.'s);

Ecosystems

Ecosystem design **instead of** platform design. Software ecosystems are an effective and economical way to construct large software systems for Industry 4.0 on top of a software platform by adding up software modules developed by different actors. In ecosystem design software engineering is spread outside the traditional borders of software companies to a group of companies and private persons.

Company 1:

- On-board systems (14 Ph.D.'s);
- Simulation systems (10 Ph.D.'s);
- Mission critical enterprise systems (14 Ph.D.'s);
- Mission critical systems of systems (10 Ph.D.'s);

Company 2:

- Avionics systems (15 Ph.D.'s);
- Communication systems of systems (9 Ph.D.'s);
- Mission-critical hybrid control systems (6 Ph.D.'s);
- Sensor fusion systems (3 Ph.D.'s);

Company 3:

- Intelligent platforms (8 Ph.D.'s);
- Distributed simulation (6 Ph.D.'s);
- Intelligent control systems (6 Ph.D.'s);

Company 4:

- Modernization of flight systems (11 Ph.D.'s);
- Satellite systems (14 Ph.D.'s);
- Unmanned aerial systems (17 Ph.D.'s);
- Generic fault tolerant computing (2 Ph.D.'s);
- Generic fault-removal techniques (5 Ph.D.'s);

Auto-adaptive control architectures

Auto-adaptive control architectures **instead of** architectures without any control mechanisms. To realize the monitoring and controlling activities in Industry 4.0 and to cope with the changing requirements and context, software systems are expected to be more reactive and self-adaptive. This generally requires built-in feedback control mechanisms in software. Self-adaptation can be realized at system level, subsystem level and/or at component-level. In addition, different styles can be adopted, such as single, master-slave, hierarchical and/or peer-to-peer control architectures.

Dynamic, Adaptive and Reconfigurable Systems
Overview and Prospective Vision

Mehmet Aksit
University of Twente
Departement of Computer Science
P.O. Box 217
7500 AE Enschede, The Netherlands
aksit@cs.utwente.nl

Zièd Choukair
ENST Bretagne
Département Informatique
Technopôle de l'Iroise
29285 Brest cedex France
Zied.Choukair@enst-bretagne.fr

Abstract—

Systems are more and more expected to work in dynamic environment, to deal with fluctuation of their characteristics and to guaranty functional and non-functional requirements. Systems should also keep compliant with the contracted quality of service. Moreover, when necessary, services and aspects should be added or removed on line. In this paper, we overview major approaches to deploy, reconfigure and adapt applications and underlying software platforms. These changes should be realized according to the evolving context and execution environment so that systems can stay compliant with the specifications and requirements of the application. We end this position paper with a vision on future directions and developments.

Introduction

To cope with the new demands, software systems have to evolve continuously. In addition, new systems embody a rich set of components implementing complex behavior such as distributed processing, various communication protocols, presentation, data support, fault-tolerance management, real-time support, transactional support, etc. Furthermore, the execution context of modern distributed systems is not static but fluctuates dynamically. To provide the expected functional service with the desired qualities, systems should be composable and adaptable. This requires proper decomposition of system into components with the appropriate composition operators and autonomous and dynamic self-adaptation mechanisms that take into account the evolution of the execution context and environment fluctuations. We term such systems as *auto-adaptive systems* (AAS).

The new multimedia telecom services could benefit from AAS since they can be deployed optimally on network equipments, be adapted to the available resources and be reconfigured automatically according to user's mobility, preferences, profiles and equipments.

To design AAS effectively, one has to deal with the following concerns: The first one relates to the deployment of the software on hardware platforms. This requires considering various constraints such as safety, security, liability, load balancing and performance. The second concern is dynamic reconfiguration of the system when the execution context changes to such an extend that the present configuration is not optimal anymore. The third concern is the continuous on-line adaptation of the execution to the environment. This may take place, for example, if the available resources fall below a certain threshold. Last but not least, an overall concern is to guarantee non-regression and safety when the system changes its configuration.

This paper first gives an overview on the dynamic reconfiguration and adaptation techniques. The paper will end with prospective directions and a vision on the perspectives of dynamic adaptable and reconfigurable systems. We think that binding components on-line through connectors and by using introspection and intercession is a very promising approach in realizing dynamic, adaptive and reconfigurable systems.

1. Dynamic reconfiguration

Traditionally, reconfiguration takes place during maintenance or when a new version of the system is installed. A reconfiguration process may be applied for rearranging the elements of various parts of the system, such as applications, platforms, system architectures, underlying infrastructures and management facilities. A dynamic reconfiguration process has to address the following issues:

- Structural changes: These are the changes that impact the topology of the application. Structural changes usually consist of adding or removing components or modifying connections among components.
- Geographical changes: Such changes impact the distribution of the components and their localization. Geographical changes are especially used for load

Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)
0-7695-1921-0/03 $17.00 © 2003 IEEE

Distributed problem solving

Distributed problem solving including distributed algorithms, coordinating systems and multi-agent architectures **instead of** centralized problem solving with monolithic and/or localized architectures. Since computer systems for Industry 4.0 are distributed, to reduce complexity and enhance reliability/availability, algorithms and intelligence in systems must be distributed as well. Accordingly, programming languages and techniques must adequately support distributed programming efforts by offering expressive and flexible abstractions.

Model-based development

Model-based development **instead of** straight-forward programming. Since more and more companies are specialized in certain product categories and in manufacturing processes, deriving software architecture from relevant domain models can help in reducing complexity, enhancing reuse and testability/verifiability of software systems. Model-based development has been adopted in various approaches such as product-line engineering (SPLE), model-driven engineering (MDE), domain specific architectures (DSA) and domain-specific programming languages (DSL), model-based verification (MBV).

Deriving Object-Oriented Frameworks From Domain Knowledge

Mehmet Aksit[1], Bedir Tekinerdogan[1], Francesco Marcelloni[2] and Lodewijk Bergmans[1, 3]

[1]TRESE Project, Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.
email: {aksit | bedir | bergmans }@cs.utwente.nl
www server: http://wwwtrese.cs.utwente.nl
[2]Department of Information Engineering, University of Pisa, Via Diotisalvi, 2-56126, Pisa, Italy.
email: france@iet.unipi.it
[3] STEX bv, D.Dijkhuisstraat 248, 7558 GG, Hengelo, The Netherlands

Abstract

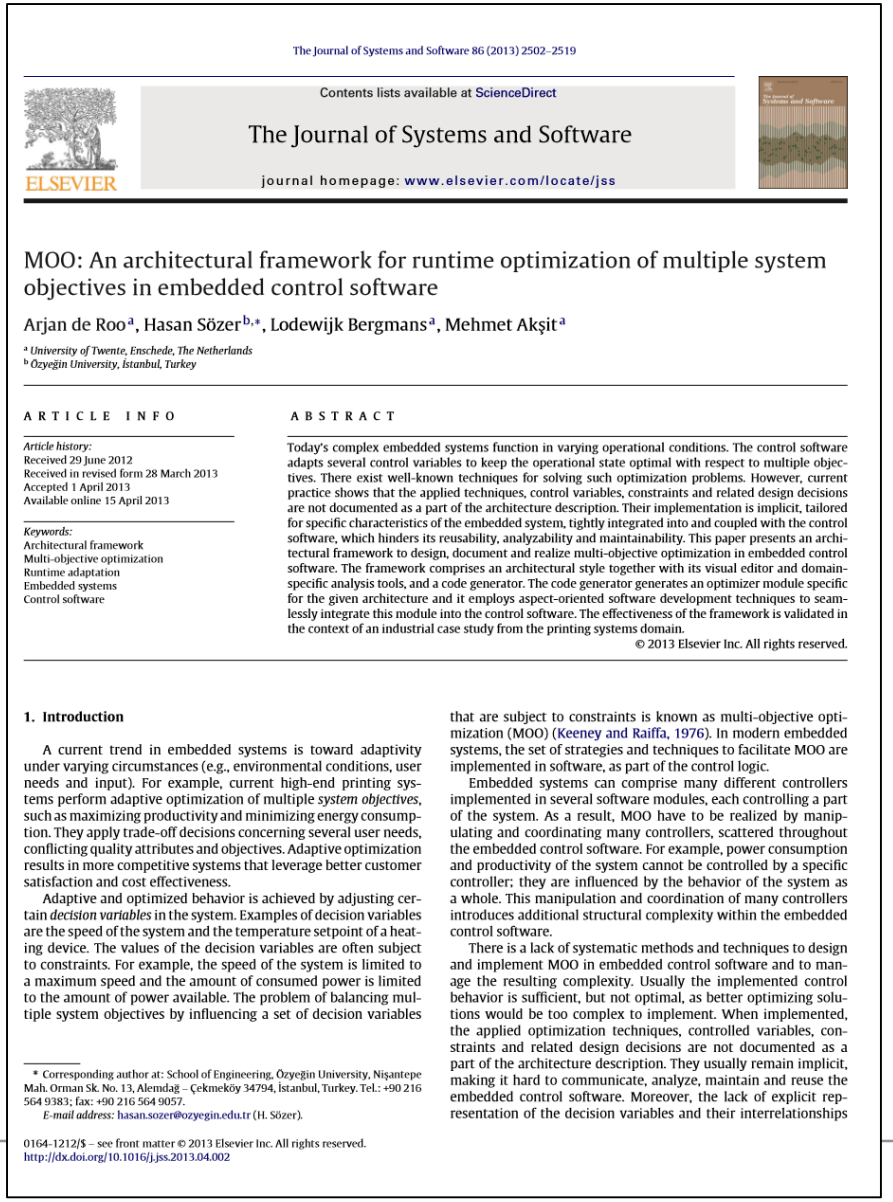
Although a considerable number of successful frameworks have been developed during the last decade, designing a high-quality framework is still a difficult task. Generally, it is assumed that finding the correct abstractions is very hard, and therefore a successful framework can only be developed through a number of iterative (software) development efforts. Accordingly, existing framework development practices span a considerable amount of refinement time, and it is worthwhile to shorten this effort. To this end, this paper aims at defining explicit models for the *knowledge domains* that are related to a framework. The absence of such models may be the main reason for the currently experienced extensive refinement effort. The applicability of the approach is illustrated by means of three pilot projects. We experienced that some aspects of domain knowledge could not be directly modeled in terms of object-oriented concepts. In this paper we describe our approach, the pilot projects, the experienced problems and the adopted solutions for realizing the frameworks. We conclude the paper with the lessons that we learned from this experience.

Correspondence address: Mehmet Aksit, TRESE Project, Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.

Published by Wiley

Multi-objective optimization

Multi-objective optimization **instead of** ad-hoc hand-crafted and/or single objective optimization. Along the line of model-based development, various algorithmic techniques and search-based methods have been introduced to compute the “optimal” architectural decomposition with respect to certain quality attributes. In addition, various run-time optimization techniques can be adopted in computing optimal control strategies and scheduling processes.



The Journal of Systems and Software 86 (2013) 2502–2519

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

MOO: An architectural framework for runtime optimization of multiple system objectives in embedded control software

Arjan de Roo[a], Hasan Sözer[b,*], Lodewijk Bergmans[a], Mehmet Akşit[a]

[a] University of Twente, Enschede, The Netherlands
[b] Özyeğin University, İstanbul, Turkey

ARTICLE INFO

Article history:
Received 29 June 2012
Received in revised form 28 March 2013
Accepted 1 April 2013
Available online 15 April 2013

Keywords:
Architectural framework
Multi-objective optimization
Runtime adaptation
Embedded systems
Control software

ABSTRACT

Today's complex embedded systems function in varying operational conditions. The control software adapts several control variables to keep the operational state optimal with respect to multiple objectives. There exist well-known techniques for solving such optimization problems. However, current practice shows that the applied techniques, control variables, constraints and related design decisions are not documented as a part of the architecture description. Their implementation is implicit, tailored for specific characteristics of the embedded system, tightly integrated into and coupled with the control software, which hinders its reusability, analyzability and maintainability. This paper presents an architectural framework to design, document and realize multi-objective optimization in embedded control software. The framework comprises an architectural style together with its visual editor and domain-specific analysis tools, and a code generator. The code generator generates an optimizer module specific for the given architecture and it employs aspect-oriented software development techniques to seamlessly integrate this module into the control software. The effectiveness of the framework is validated in the context of an industrial case study from the printing systems domain.

1. Introduction

A current trend in embedded systems is toward adaptivity under varying circumstances (e.g., environmental conditions, user needs and input). For example, current high-end printing systems perform adaptive optimization of multiple *system objectives*, such as maximizing productivity and minimizing energy consumption. They apply trade-off decisions concerning several user needs, conflicting quality attributes and objectives. Adaptive optimization results in more competitive systems that leverage better customer satisfaction and cost effectiveness.

Adaptive and optimized behavior is achieved by adjusting certain *decision variables* in the system. Examples of decision variables are the speed of the system and the temperature setpoint of a heating device. The values of the decision variables are often subject to constraints. For example, the speed of the system is limited to a maximum speed and the amount of consumed power is limited to the amount of power available. The problem of balancing multiple system objectives by influencing a set of decision variables that are subject to constraints is known as multi-objective optimization (MOO) (Keeney and Raiffa, 1976). In modern embedded systems, the set of strategies and techniques to facilitate MOO are implemented in software, as part of the control logic.

Embedded systems can comprise many different controllers implemented in several software modules, each controlling a part of the system. As a result, MOO have to be realized by manipulating and coordinating many controllers, scattered throughout the embedded control software. For example, power consumption and productivity of the system cannot be controlled by a specific controller; they are influenced by the behavior of the system as a whole. This manipulation and coordination of many controllers introduces additional structural complexity within the embedded control software.

There is a lack of systematic methods and techniques to design and implement MOO in embedded control software and to manage the resulting complexity. Usually the implemented control behavior is sufficient, but not optimal, as better optimizing solutions would be too complex to implement. When implemented, the applied optimization techniques, controlled variables, constraints and related design decisions are not documented as a part of the architecture description. They usually remain implicit, making it hard to communicate, analyze, maintain and reuse the embedded control software. Moreover, the lack of explicit representation of the decision variables and their interrelationships

* Corresponding author at: School of Engineering, Özyeğin University, Nişantepe Mah. Orman Sk. No. 13, Alemdağ – Çekmeköy 34794, İstanbul, Turkey. Tel.: +90 216 564 9383; fax: +90 216 564 9057.
E-mail address: hasan.sozer@ozyegin.edu.tr (H. Sözer).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.04.002

Modularization of semantic concerns

Modularization of semantic concerns **instead of** traditional abstraction mechanisms based on implementation concerns such as data or function. As a consequence of model-based development, software abstractions more and more correspond to the concerns of models. The concerns of a model are naturally based on the semantics of the model, and these cannot always be effectively represented as a data or function. Moreover, concerns in Industry 4.0 systems can be emerging meaning that they may appear or disappear dynamically. As such, programming languages and techniques must adequately support programming efforts by offering expressive and flexible abstractions for emergent semantic concerns.

On Liberating Programs from the von Neumann Architecture via Event-Based Modularization

Somayeh Malakuti *
Software Technology group
Technical University of Dresden, Germany
somayeh.malakuti@tu-dresden.de

Mehmet Aksit
Software Technology group
University of Twente, the Netherlands
m.aksit@utwente.nl

Abstract

From the early days of computers, researchers have been trying to invent effective and efficient means for expressing software systems through the introduction of new programming languages. In the early days, due to the limitations of the technology, the abstractions of the programming languages were conceptually close to the abstractions of the von Neumann based realization platforms. With the advancement of the technology, computers have been increasingly applied for complex problems in different application domains. This required the challenge of designing programming languages that resemble more the semantics of software rather than the concepts of underlying machinery. To this aim, various new language concepts, such as object-oriented, aspect-oriented, and event-based languages have been introduced. While these languages were successful in enhancing the expression power of languages towards more semantic concerns of application domains, they fail in short in representing emergent behavioral patterns of software effectively. We outline a set of requirements to overcome these shortcomings, and explain the concept of **event-based modularization** as a possible solution.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features—Modules, packages

General Terms Languages, Design

Keywords Emergent behavior, modularity, von Neumann architecture

1. Introduction

Although there are many facets of software engineering, its main objective is to create software systems that execute on hardware/-software platforms [3, p.9-12]. From the early days of computers, researchers have been trying to invent effective and efficient means for expressing software systems through the introduction of new programming languages.

Along this line, we observe a continuous interplay between programming languages and platforms. We claim that to create better languages, one needs to understand the historical nature of this interplay and the forces that influence it. From this perspective, we will make an attempt to characterize platforms, programming languages and the constraints among these.

During the last decades platforms have evolved dramatically. They are now many magnitudes more powerful than their previous generations, they consist of multicores and they are based on geographically distributed networked architectures. Despite these developments, at their core, platforms still display typical characteristics of von Neumann architecture of instructions, registers and memory locations. In the literature [1, 5], programming languages are generally classified based on the paradigms that they adopt. Functional, logic, and imperative languages are considered as the three basic categories; they all have equivalent expression power in the sense that any executable program can be expressed in one of these categories.

In practice, however, the main reason why a programming language might be selected is more based on the pragmatic factors such as availability of the tools, programmers' skills and on the desired non-functional characteristics of programs such as adaptability, flexibility and evolvability rather than the paradigm of the language. Moreover, most commercial languages are not based on a single paradigm but they borrow features from multiple paradigms. Imperative languages have been much more successful in practice and therefore in this paper we will mainly focus on these languages.

Every language provides a set of first-class abstractions that are directly supported by the mechanisms of that language. For example, a first-class abstraction may be passed as an argument of a call, it may be returned as a result of a call, it may be stored or retrieved, etc. The interplay between languages and platforms is defined by the following constraint [1]: the first-class abstractions of a language as much as possible must match natural abstractions and semantics of the considered problem domain. On the other hand, they must be concrete enough to compile them easily and efficiently onto the available platforms.

With the enormous increase in the complexity of software, the concept of separation of concerns have become more and more important. Naturally, the first class abstractions of the languages have become the essential way of expressing the separation of concerns in programs. They become important factors in evaluating languages because they define the direct support that a programming language offers for expressing concerns in programs.

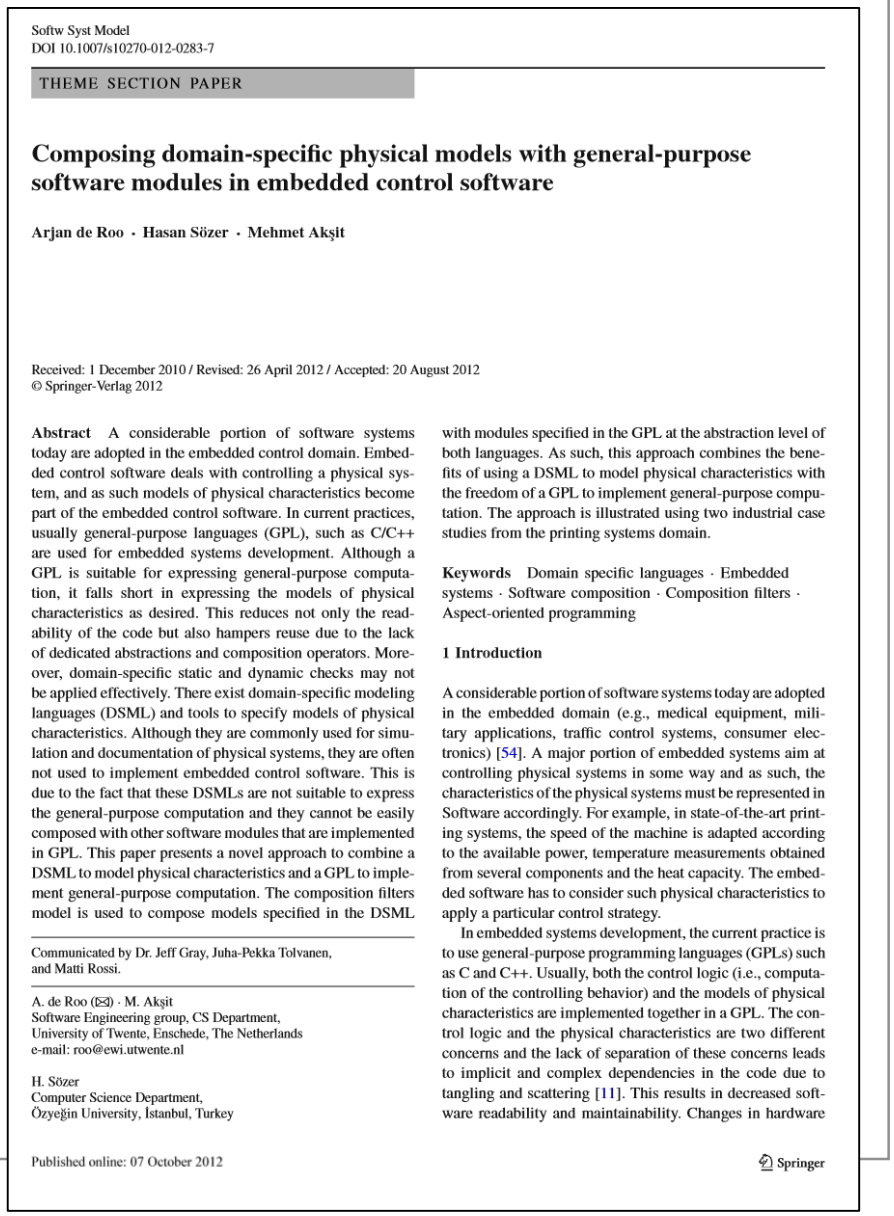
In the following sections, to analyze the programming languages historically, we will characterize them with respect to their first-class abstractions, and we will historically categorize the de-

* The author is supported by the German Research Foundation (DFG) in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing".

MODULARITY Companion'15, March 16–19, 2015, Fort Collins, CO, USA
Copyright 2015 ACM 978-1-4503-3283-5/15/03...$15.00
http://dx.doi.org/10.1145/2735386.2735387

A rich set of composition mechanisms

A rich set of composition mechanisms **instead of** a fixed set of language constructs for hierarchical organization of programs (such as class-inheritance). To support flexibility in control strategies and to cope with various evolution schemes, languages must offer generic and/or domain specific composition mechanisms to express, for example, object, aspect and event compositions and transformational techniques in a uniform manner. The languages must maintain their closure property in compositions so that scale-ability of systems can be provided.



Softw Syst Model
DOI 10.1007/s10270-012-0283-7

THEME SECTION PAPER

Composing domain-specific physical models with general-purpose software modules in embedded control software

Arjan de Roo · Hasan Sözer · Mehmet Akşit

Received: 1 December 2010 / Revised: 26 April 2012 / Accepted: 20 August 2012

Abstract A considerable portion of software systems today are adopted in the embedded control domain. Embedded control software deals with controlling a physical system, and as such models of physical characteristics become part of the embedded control software. In current practices, usually general-purpose languages (GPL), such as C/C++ are used for embedded systems development. Although a GPL is suitable for expressing general-purpose computation, it falls short in expressing the models of physical characteristics as desired. This reduces not only the readability of the code but also hampers reuse due to the lack of dedicated abstractions and composition operators. Moreover, domain-specific static and dynamic checks may not be applied effectively. There exist domain-specific modeling languages (DSML) and tools to specify models of physical characteristics. Although they are commonly used for simulation and documentation of physical systems, they are often not used to implement embedded control software. This is due to the fact that these DSMLs are not suitable to express the general-purpose computation and they cannot be easily composed with other software modules that are implemented in GPL. This paper presents a novel approach to combine a DSML to model physical characteristics and a GPL to implement general-purpose computation. The composition filters model is used to compose models specified in the DSML with modules specified in the GPL at the abstraction level of both languages. As such, this approach combines the benefits of using a DSML to model physical characteristics with the freedom of a GPL to implement general-purpose computation. The approach is illustrated using two industrial case studies from the printing systems domain.

Keywords Domain specific languages · Embedded systems · Software composition · Composition filters · Aspect-oriented programming

Communicated by Dr. Jeff Gray, Juha-Pekka Tolvanen, and Matti Rossi.

A. de Roo (✉) · M. Akşit
Software Engineering group, CS Department,
University of Twente, Enschede, The Netherlands
e-mail: roo@ewi.utwente.nl

H. Sözer
Computer Science Department,
Özyeğin University, İstanbul, Turkey

Published online: 07 October 2012

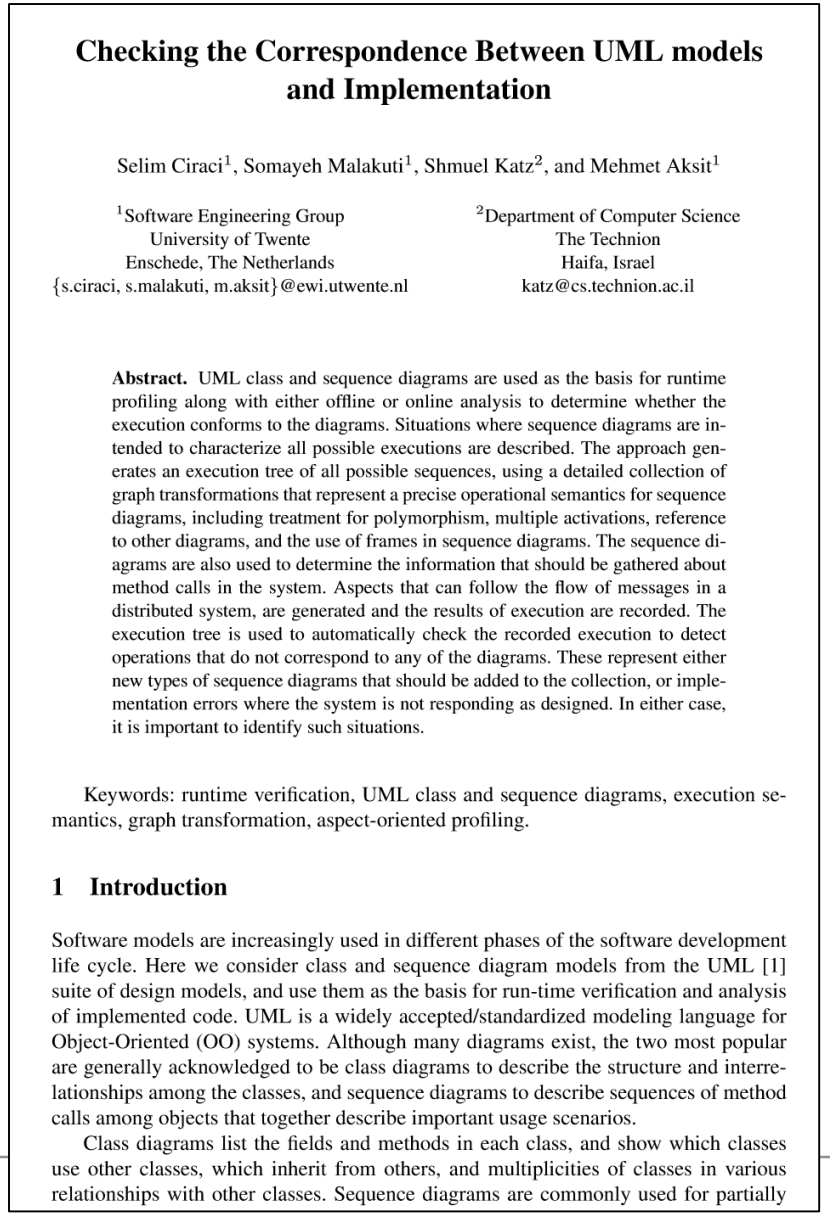
1 Introduction

A considerable portion of software systems today are adopted in the embedded domain (e.g., medical equipment, military applications, traffic control systems, consumer electronics) [54]. A major portion of embedded systems aim at controlling physical systems in some way and as such, the characteristics of the physical systems must be represented in Software accordingly. For example, in state-of-the-art printing systems, the speed of the machine is adapted according to the available power, temperature measurements obtained from several components and the heat capacity. The embedded software has to consider such physical characteristics to apply a particular control strategy.

In embedded systems development, the current practice is to use general-purpose programming languages (GPLs) such as C and C++. Usually, both the control logic (i.e., computation of the controlling behavior) and the models of physical characteristics are implemented together in a GPL. The control logic and the physical characteristics are two different concerns and the lack of separation of these concerns leads to implicit and complex dependencies in the code due to tangling and scattering [11]. This results in decreased software readability and maintainability. Changes in hardware

Uniform integration of verification techniques

Uniform integration of verification techniques **instead of** independent tool- and technique-specific verification approaches. There are various model-based verification and testing approaches available. Examples are model-checking, static and dynamic analysis, run-time verification, model-based testing, adopting model-specific verification (simultaneously) based on continuous and/or discrete models, etc. Most of these techniques are complementary and as such combined usage of these may help in finding faults with less false-positive and false-negative cases.



Checking the Correspondence Between UML models and Implementation

Selim Ciraci[1], Somayeh Malakuti[1], Shmuel Katz[2], and Mehmet Aksit[1]

[1]Software Engineering Group
University of Twente
Enschede, The Netherlands
{s.ciraci, s.malakuti, m.aksit}@ewi.utwente.nl

[2]Department of Computer Science
The Technion
Haifa, Israel
katz@cs.technion.ac.il

Abstract. UML class and sequence diagrams are used as the basis for runtime profiling along with either offline or online analysis to determine whether the execution conforms to the diagrams. Situations where sequence diagrams are intended to characterize all possible executions are described. The approach generates an execution tree of all possible sequences, using a detailed collection of graph transformations that represent a precise operational semantics for sequence diagrams, including treatment for polymorphism, multiple activations, reference to other diagrams, and the use of frames in sequence diagrams. The sequence diagrams are also used to determine the information that should be gathered about method calls in the system. Aspects that can follow the flow of messages in a distributed system, are generated and the results of execution are recorded. The execution tree is used to automatically check the recorded execution to detect operations that do not correspond to any of the diagrams. These represent either new types of sequence diagrams that should be added to the collection, or implementation errors where the system is not responding as designed. In either case, it is important to identify such situations.

Keywords: runtime verification, UML class and sequence diagrams, execution semantics, graph transformation, aspect-oriented profiling.

1 Introduction

Software models are increasingly used in different phases of the software development life cycle. Here we consider class and sequence diagram models from the UML [1] suite of design models, and use them as the basis for run-time verification and analysis of implemented code. UML is a widely accepted/standardized modeling language for Object-Oriented (OO) systems. Although many diagrams exist, the two most popular are generally acknowledged to be class diagrams to describe the structure and interrelationships among the classes, and sequence diagrams to describe sequences of method calls among objects that together describe important usage scenarios.

Class diagrams list the fields and methods in each class, and show which classes use other classes, which inherit from others, and multiplicities of classes in various relationships with other classes. Sequence diagrams are commonly used for partially

Conclusions

Conclusions

- The trends in industry cannot be controlled! Therefore they must be understood and managed, as much as possible;
- We have presented a unique set of features of these trends and position these w.r.t. Industry 4.0;
- Define a company maturation process to help companies to deal with the trends;
- We have applied this process to 4 large companies;
- We have evaluated these w.r.t. to the trends in CS and present the expected challenges/topics in software engineering and technology.