TOPICS FOR THESES AND PRACTICAL PROJECTS WS 2022

Institute for Software Systems Engineering

http://www.isse.jku.at

JOHANNES KEPLER UNIVERSITÄT LINZ

Institute for Software Systems Engineering
Technologies for building better software

TOPICS

- We have many topics

- Contact the person in charge for appointment
- It is possible to do teamwork
- It is also possible for two to choose the same topic
- Languages: German and English

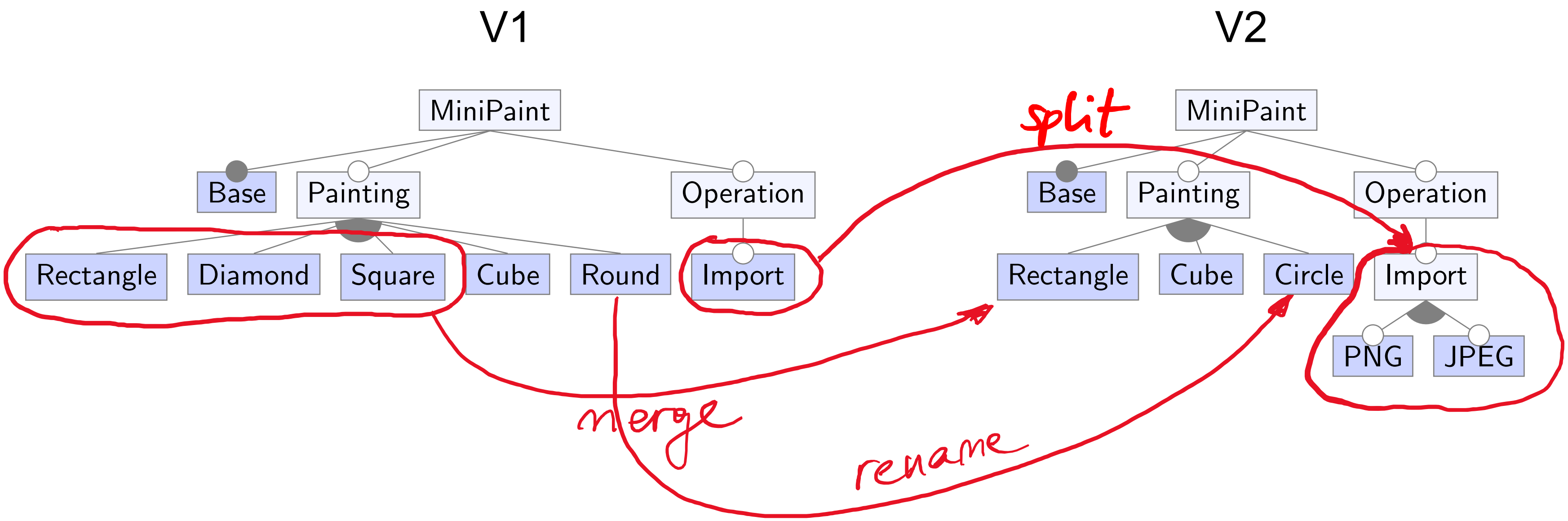
MEETINGS

- Four Joint Meetings
  - Assignment and Definition of Topics
  - Two Status Reports with Intermediate Results
  - Final Report with a Short Demo
- Regular Individual Meetings with Advisors

Datum	Uhrzeit	Raum
Do. 13.10.2022	10:00 - 11:30	S3 218
Do. 10.11.2022	09:00 - 10:30	S3 218
Do. 15.12.2022	09:00 - 10:30	S3 218
Do. 19.01.2023	09:00 - 10:30	S3 218

Product Line Refactoring

PRODUCT LINES EVOLVE



- Adding new features
- Revising existing features
- Merging features
- Splitting features
- …

Challenge: Keeping feature-to-code mappings up to date

- Particularly hard when manually creating and maintaining them
- E.g., annotation-based PLs
  [Michelon2020, Michelon2021]

JKU

IDEA: COMBINING THE BENEFITS OF GIT AND ECCO

Git

- Proven and widely used
- Huge ecosystem of tools
- Useful in many different workflows

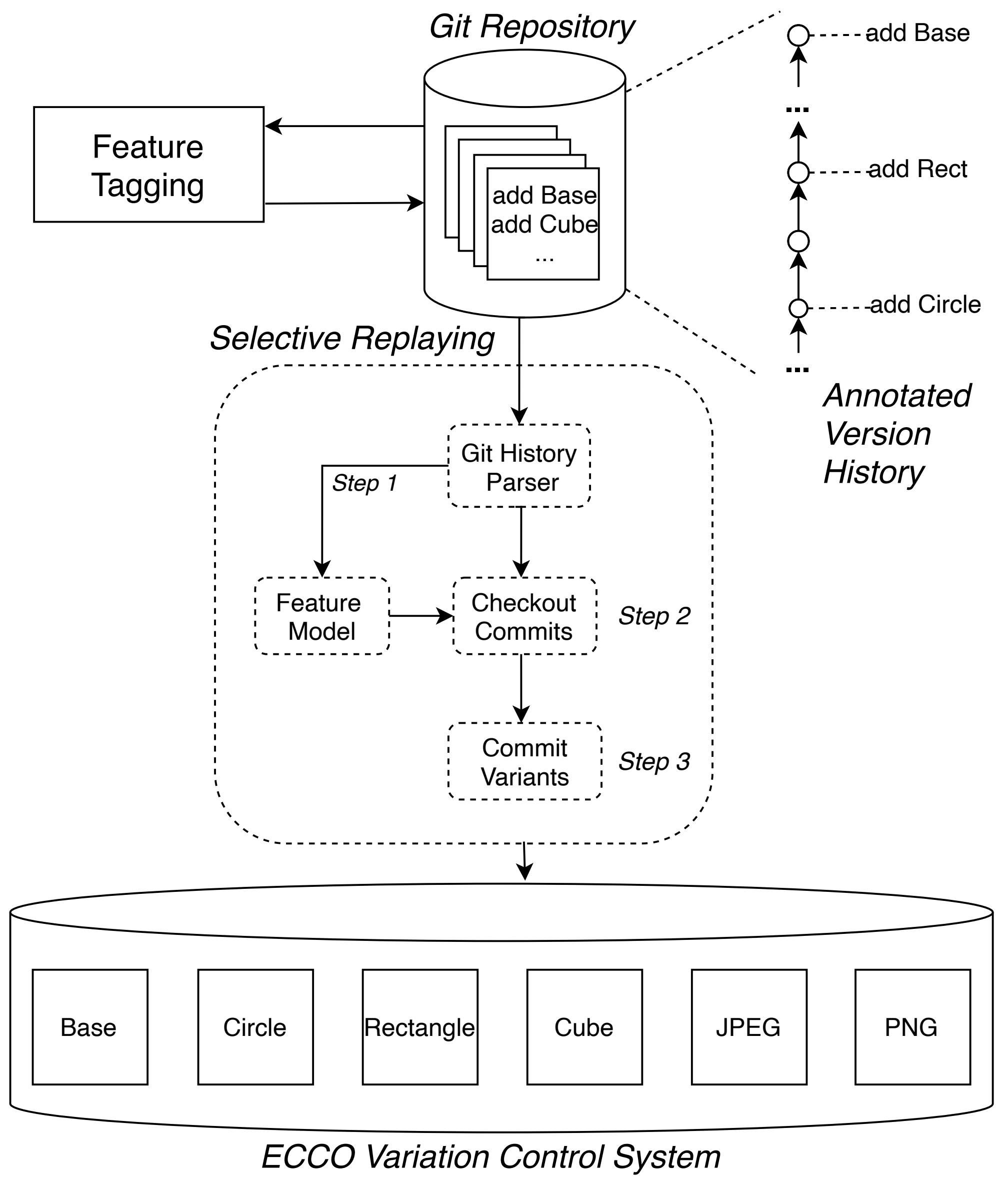
ECCO

- Feature-to-code mappings created automatically
- Handling fine-grained revisions and variants
- Composing new versions based on features

Potential Use Cases

- Migrating existing systems to product lines
- Refactoring features of existing product lines

RESERVE TOOL



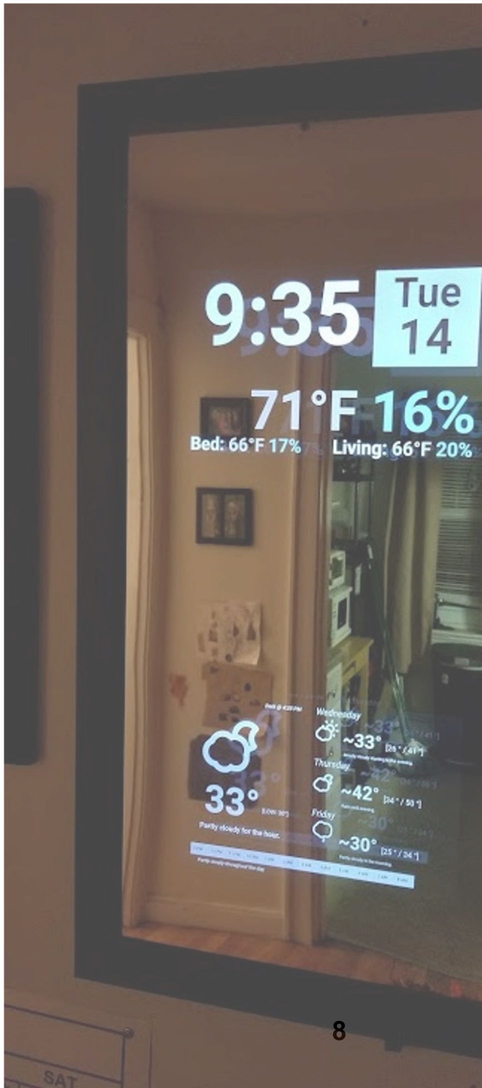
PRELIMINARY EVALUATION

Git Repository of a Magic Mirror System

- ☐ Written in Java
  - ~ 50 Files
  - ~ 2,100 LOC
- ☐ 100 Git commits
- ☐ Configuration Files: JSON, Text, XML
- ☐ Static resources: JSON
- ☐ SpringBoot Application
- ☐ Modular

Features

- ☐ Base (mand.)
- ☐ Traffic
- ☐ Weather
- ☐ I18N
- ☐ Settings
- ☐ RSS



TOPIC: RESERVE CASE STUDY TOPICS

- Apply ReSerVe on a larger system
- Select Java-based system
- Analyse evolution history
- Perform replay experiments
- Process described in existing conference paper

Refactoring Product Lines by Replaying Version Histories

Michael Ratzenböck
Paul Grünbacher
Wesley Klewerton Guez Assunção
Alexander Egyed
Institute of Software Systems Engineering
Johannes Kepler University Linz
4040 Linz, Austria
paul.gruenbacher@jku.at

Lukas Linsbauer
Institute of Software Engineering and Automotive Informatics
Technische Universität Braunschweig
38106 Braunschweig, Germany
l.linsbauer@tu-braunschweig.de

ABSTRACT

When evolving software product lines, new features are added over time and existing features are revised. Engineers also decide to merge different features or split features in other cases. Such refactoring tasks are difficult when using manually maintained feature-to-code mappings. Intensional version control systems such as ECCO overcome this issue with automatically computed feature-to-code mappings. Furthermore, they allow creating variants that have not been explicitly committed before. However, such systems are still rarely used compared to extensional version control systems like Git, which keep track of the evolution history by assigning revisions to states of a system. This paper presents an approach combining both extensional and intensional version control systems, which relies on the extensional version control system Git to store versions. Developers selectively tag existing versions to describe the evolution at the level of features. Our approach then automatically replays the evolution history to create a repository of the intensional variation control system ECCO. The approach contributes to research on refactoring features of existing product lines and migrating existing systems to product lines. We provide an initial evaluation of the approach regarding correctness and performance based on an existing system.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Software configuration management and version control systems**; *Software maintenance tools.*

KEYWORDS

version control systems, refactoring, feature-level evolution

ACM Reference Format:
Michael Ratzenböck, Paul Grünbacher, Wesley Klewerton Guez Assunção, Alexander Egyed, and Lukas Linsbauer. 2022. Refactoring Product Lines by Replaying Version Histories. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS '22), February 23–25, 2022, Florence, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510466.3510484

VAMOS '22, February 23–25, 2022, Florence, Italy

ACM ISBN 978-1-4503-9604-2/22/02…$15.00
https://doi.org/10.1145/3510466.3510484

1 INTRODUCTION

Product lines are subject to continuous evolution [13, 23]. Features are added, removed, and renamed over time, and they are split or merged to accommodate new or changing requirements [3]. These continuous changes result in many revisions of software artifacts [7]. Revisions are the result of evolution in time, e.g., when fixing a bug. They denote sequential versions, representing a snapshot of the evolution of a software feature. Variants on the other hand stem from evolution in space [2], e.g., when adding a new feature. They denote versions of software artifacts that need to exist concurrently. In annotation-based product lines engineers manually maintain feature-to-code mappings. Maintaining code fragments guarded by annotations encoding the mappings is hard [15, 21] and it is particularly challenging to carry out changes to features while at the same time keeping the mappings consistent [13, 22, 28]. For instance, merging features at a certain point is difficult when done manually, since features are mapped to diverse and complex artifacts.

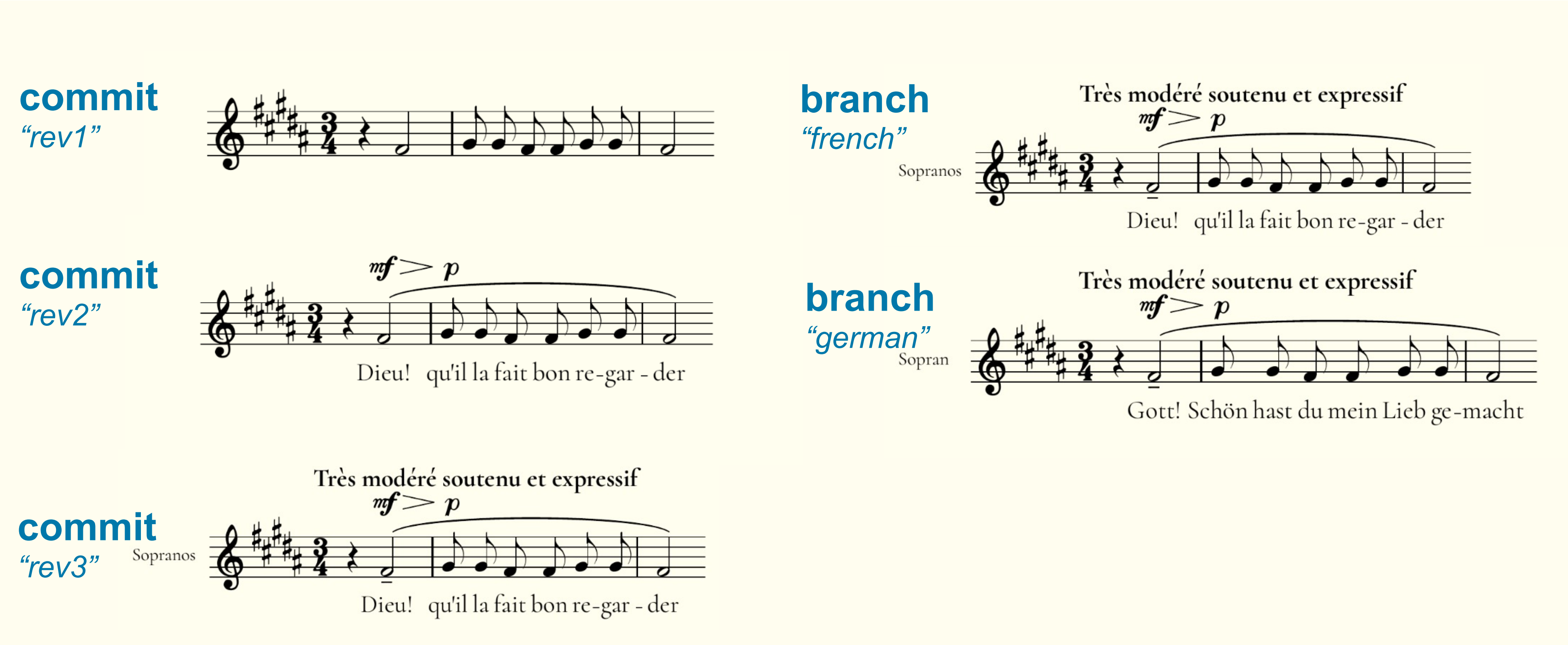
Existing version control systems pursue two versioning strategies [7, 18], which can be used to manage evolving product lines: *Extensional versioning* assumes that all existing versions are explicitly enumerated. It then allows to retrieve the versions that have been created before. Git or Subversion are examples of such tools, which keep track of changes by assigning revisions to states of a system over time. However, evolution is rarely just a linear sequence of steps and such tools thus provide branching mechanisms for dealing with variants. For instance, short-term branches are used to develop new features in isolation. Once a new feature is finished, it is merged with the original artifact and the branch is no longer used. However, at this point the new feature becomes tangled with the rest of the artifacts and its location is not managed explicitly [22]. The purpose of long-term branches, on the other hand, is to create clones of existing artifacts, based on which variants of the system can then be created. Nonetheless, long-term branches quickly lead to maintenance problems as updates and fixes need to be propagated to all variants [25]. *Intensional versioning* aims at overcoming these limitations with mechanisms for managing fine-grained variants [18], thereby avoiding branches for features of variants. Furthermore, they allow creating versions that have not been explicitly enumerated and committed before. Such tools use concepts like features, configurations, and construction

For this topic please contact: paul.gruenbacher@jku.at

Intensional Versioning Study

EXTENSIONAL VCS (E.G., GIT, SUBVERSION)

Revisions: *Evolution in Time* **Variants**: *Evolution in Space*



- Use **branches** for coarse-grained and **annotations** for fine-grained variants
- **Manual** updates of feature-to-code mappings
- Can retrieve only **previously committed versions** (e.g., rev2, french)

INTENSIONAL VCS (E.G., ECCO, SUPERMOD)

Revisions: *Evolution in Time*

commit
+setup.1
+notes.1

commit
+slurs.1
+dynamics.1
+lyrics.1

commit
+texts.1
+articulations.1

Variants: *Evolution in Space*

commit
+french.1

commit
+german.1

checkout *setup.1, notes.1, dynamics.1*

- Uniform handling of revisions and variants (a.k.a. Variation Control Systems)
- Committing **features** with **automatic updates** of feature-to-code mappings
- Can compose **new versions** based on **features** of committed versions

[Linsbauer et al., 2021]

TOPIC: STUDY ON INTENSIONAL VERSIONING AND CORRECTNESS

Correctness Levels

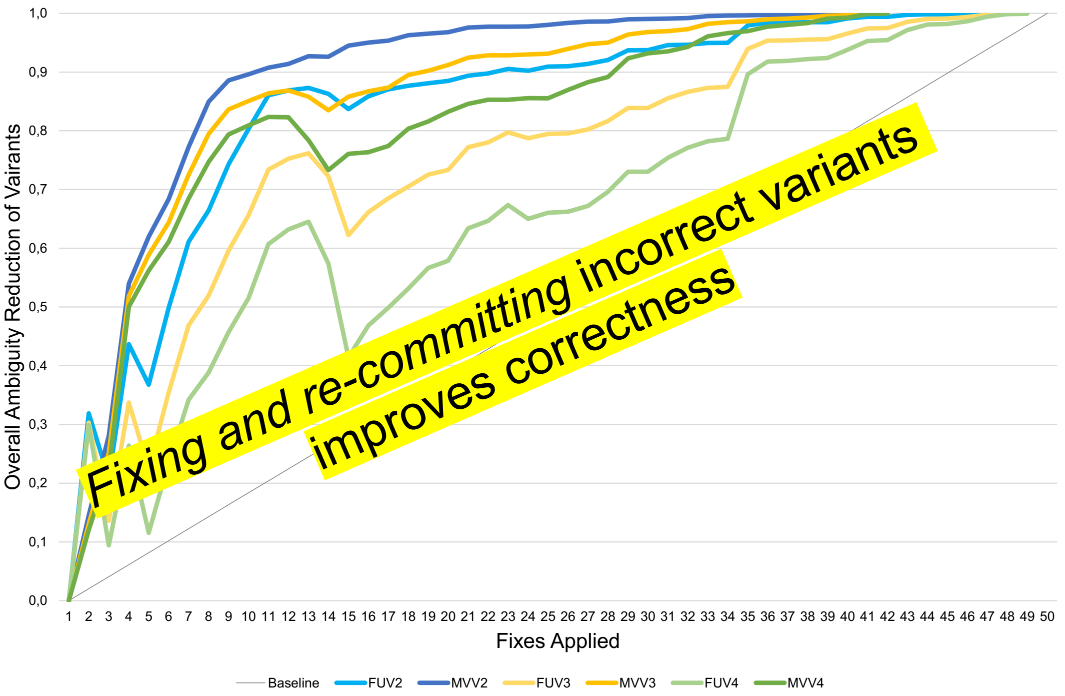
CL5: Compiles and runs

CL4: Compiles and runs after removing surplus code

CL3: Compiles and runs with runtime errors after removing surplus code

CL2: Compilation errors even after removing surplus code

CL1: Required code is missing or features cannot be distinguished



For this topic please contact: paul.gruenbacher@jku.at

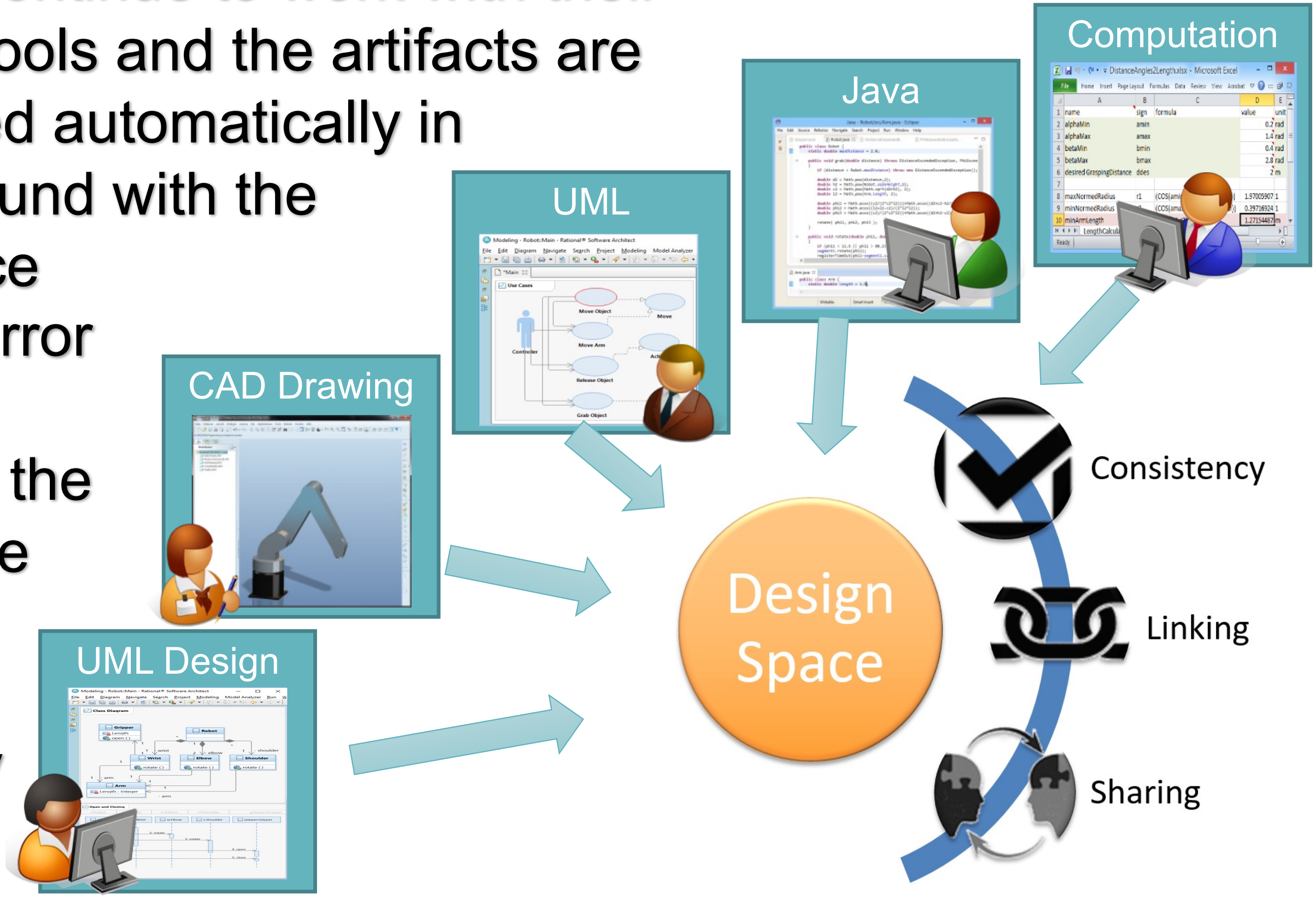
- Empirical Study: Impact of fixing and committing incorrect variants on correctness levels
- Possible Artifacts: Java Code, Lilypond Music DSL, etc.

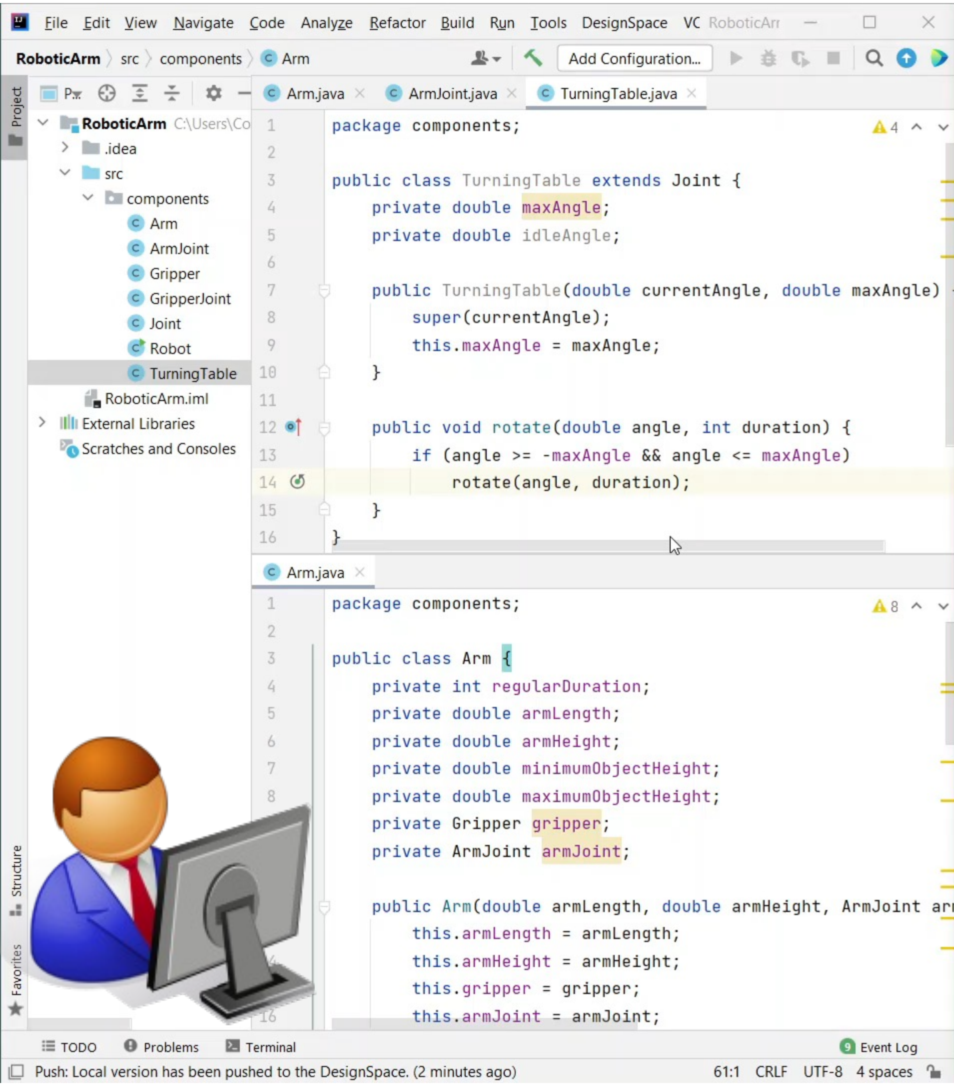
DesignSpace

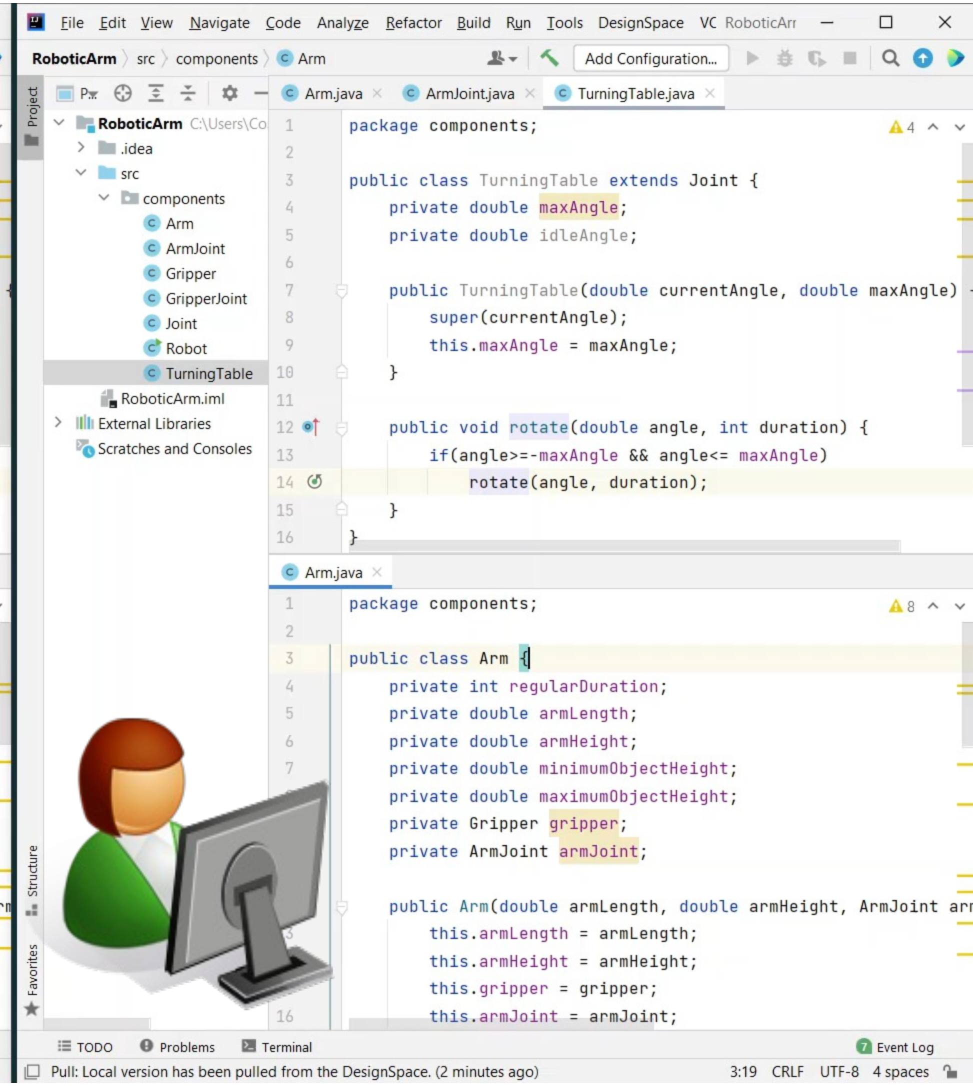
LIT Exzellenzprojekt

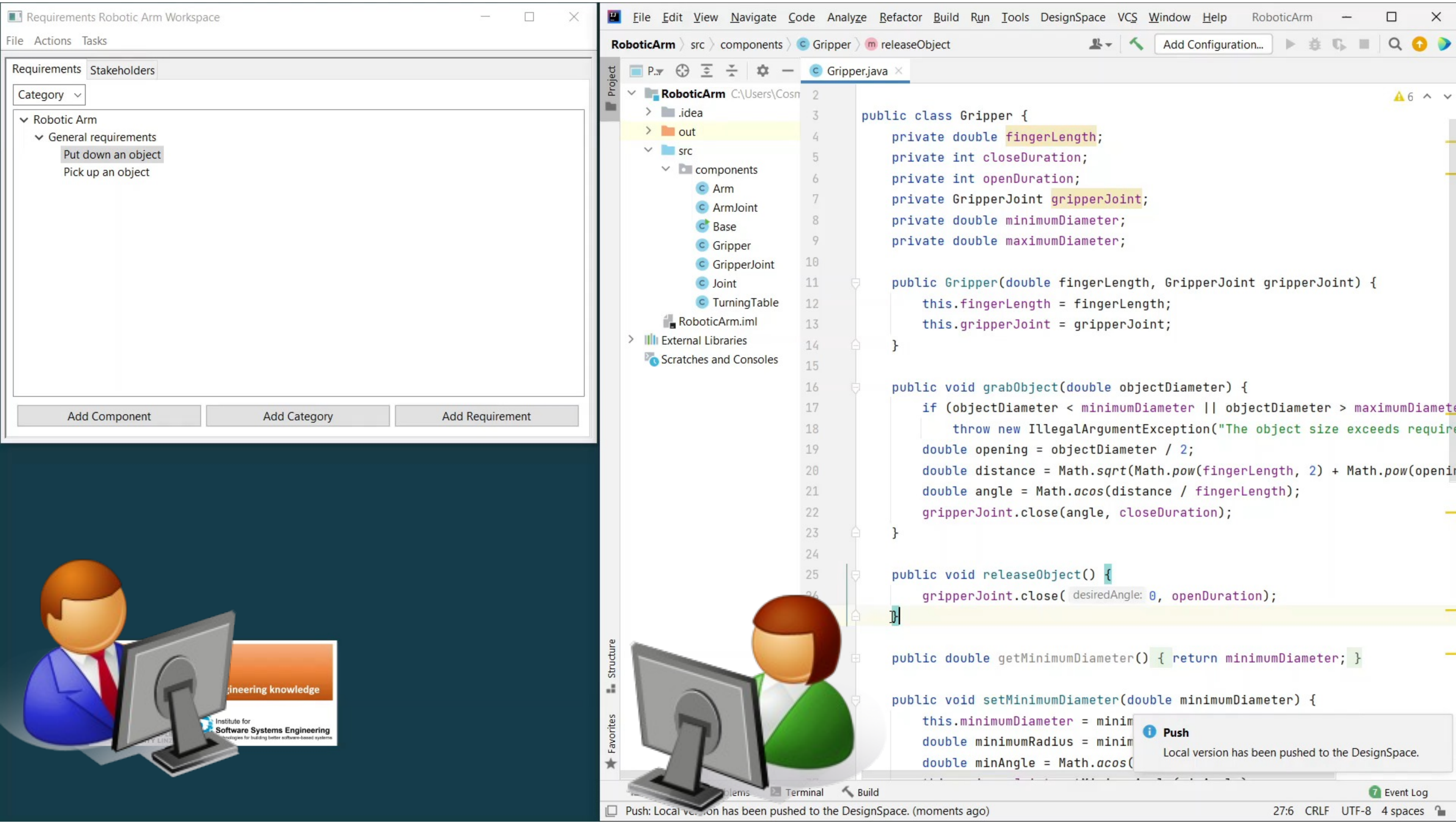
DESIGNSPACE

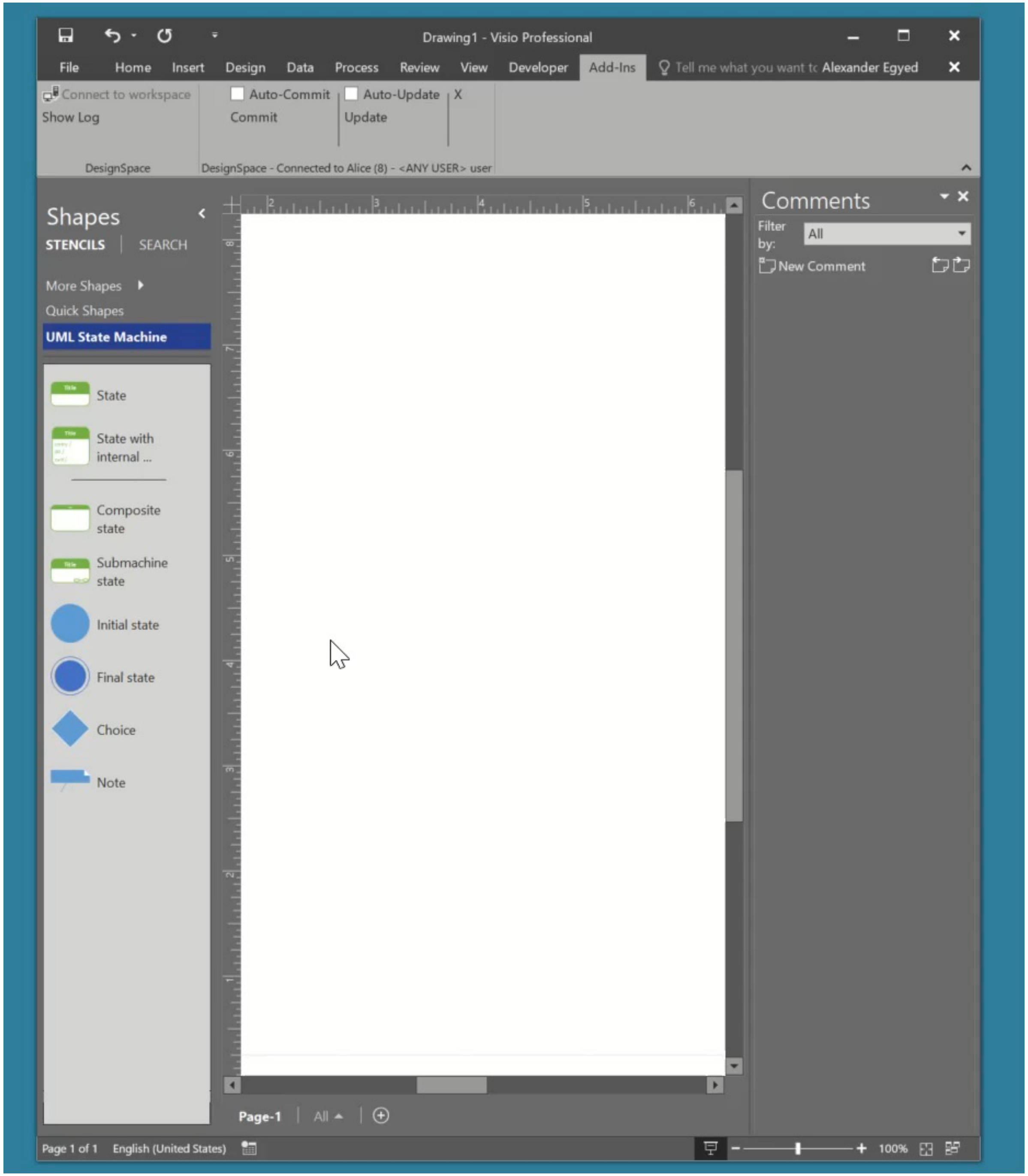
- Engineers continue to work with their respective tools and the artifacts are synchronized automatically in the background with the DesignSpace
- Linking or Error Detection provided by the Designspace support engineers in their daily work




File Edit View Navigate Code Analyze Refactor Build Run Tools DesignSpace VC RoboticArr
RoboticArm src components Arm
Add Configuration...
Arm.java ArmJoint.java TurningTable.java
RoboticArm C:\Users\Co
.idea
src
components
Arm
ArmJoint
Gripper
GripperJoint
Joint
Robot
TurningTable
RoboticArm.iml
External Libraries
Scratches and Consoles
package components;
public class TurningTable extends Joint {
private double maxAngle;
private double idleAngle;
public TurningTable(double currentAngle, double maxAngle) {
super(currentAngle);
this.maxAngle = maxAngle;
}
public void rotate(double angle, int duration) {
if (angle >= -maxAngle && angle <= maxAngle)
rotate(angle, duration);
}
}
package components;
public class Arm {
private int regularDuration;
private double armLength;
private double armHeight;
private double minimumObjectHeight;
private double maximumObjectHeight;
private Gripper gripper;
private ArmJoint armJoint;
public Arm(double armLength, double armHeight, ArmJoint arm
this.armLength = armLength;
this.armHeight = armHeight;
this.gripper = gripper;
this.armJoint = armJoint;
TODO Problems Terminal Event Log
Push: Local version has been pushed to the DesignSpace. (2 minutes ago)
61:1 CRLF UTF-8 4 spaces


File Edit View Navigate Code Analyze Refactor Build Run Tools DesignSpace VC RoboticArr
RoboticArm src components Arm
Add Configuration...
Arm.java ArmJoint.java TurningTable.java
RoboticArm C:\Users\Co
.idea
src
components
Arm
ArmJoint
Gripper
GripperJoint
Joint
Robot
TurningTable
RoboticArm.iml
External Libraries
Scratches and Consoles
package components;
public class TurningTable extends Joint {
private double maxAngle;
private double idleAngle;
public TurningTable(double currentAngle, double maxAngle) {
super(currentAngle);
this.maxAngle = maxAngle;
}
public void rotate(double angle, int duration) {
if(angle>=-maxAngle && angle<= maxAngle)
rotate(angle, duration);
}
}
package components;
public class Arm {
private int regularDuration;
private double armLength;
private double armHeight;
private double minimumObjectHeight;
private double maximumObjectHeight;
private Gripper gripper;
private ArmJoint armJoint;
public Arm(double armLength, double armHeight, ArmJoint arm
this.armLength = armLength;
this.armHeight = armHeight;
this.gripper = gripper;
this.armJoint = armJoint;
TODO Problems Terminal Event Log
Pull: Local version has been pulled from the DesignSpace. (2 minutes ago)
3:19 CRLF UTF-8 4 spaces


Requirements Robotic Arm Workspace
File
Actions
Tasks
Requirements
Stakeholders
Category
Robotic Arm
General requirements
Put down an object
Pick up an object
Add Component
Add Category
Add Requirement
gineering knowledge
Institute for
Software Systems Engineering
File Edit View Navigate Code Analyze Refactor Build Run Tools DesignSpace VCS Window Help RoboticArm
RoboticArm src components Gripper releaseObject
Add Configuration...
Gripper.java
RoboticArm C:\Users\Cosn
.idea
out
src
components
Arm
ArmJoint
Base
Gripper
GripperJoint
Joint
TurningTable
RoboticArm.iml
External Libraries
Scratches and Consoles
public class Gripper {
private double fingerLength;
private int closeDuration;
private int openDuration;
private GripperJoint gripperJoint;
private double minimumDiameter;
private double maximumDiameter;
public Gripper(double fingerLength, GripperJoint gripperJoint) {
this.fingerLength = fingerLength;
this.gripperJoint = gripperJoint;
}
public void grabObject(double objectDiameter) {
if (objectDiameter < minimumDiameter || objectDiameter > maximumDiamete
throw new IllegalArgumentException("The object size exceeds require
double opening = objectDiameter / 2;
double distance = Math.sqrt(Math.pow(fingerLength, 2) + Math.pow(openi
double angle = Math.acos(distance / fingerLength);
gripperJoint.close(angle, closeDuration);
}
public void releaseObject() {
gripperJoint.close(desiredAngle: 0, openDuration);
}
public double getMinimumDiameter() { return minimumDiameter; }
public void setMinimumDiameter(double minimumDiameter) {
this.minimumDiameter = minim
double minimumRadius = minim
double minAngle = Math.acos(
Push
Local version has been pushed to the DesignSpace.
Terminal
Build
Event Log
Push: Local version has been pushed to the DesignSpace. (moments ago)
27:6 CRLF UTF-8 4 spaces



Drawing1 - Visio Professional
File
Home
Insert
Design
Data
Process
Review
View
Developer
Add-Ins
Tell me what you want tc
Alexander Egyed
Connect to workspace
Show Log
Auto-Commit
Commit
Auto-Update
Update
X
DesignSpace
DesignSpace - Connected to Alice (8) - <ANY USER> user
Shapes
STENCILS
SEARCH
More Shapes
Quick Shapes
UML State Machine
State
State with internal ...
Composite state
Submachine state
Initial state
Final state
Choice
Note
Comments
Filter by:
All
New Comment
Page-1
All
Page 1 of 1
English (United States)
100%

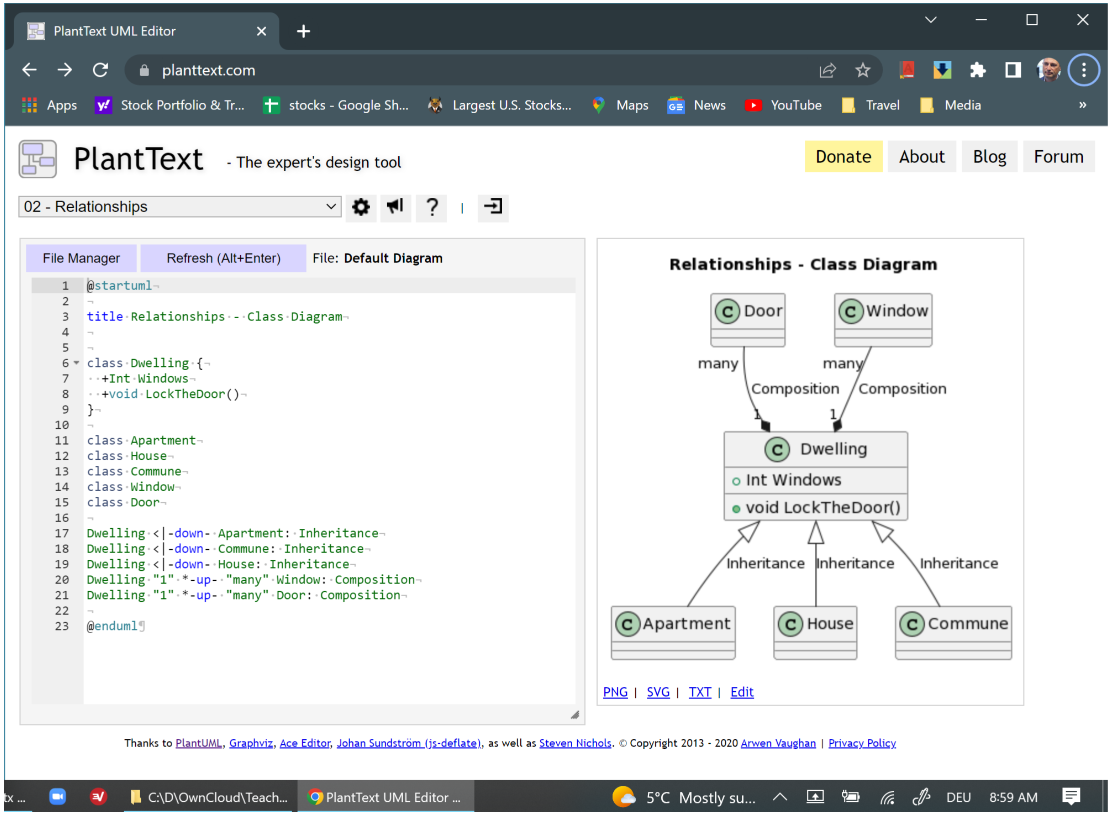
REST API
(ALEXANDER.EGYED@JKU.AT)

- Blockly and other applciations use Java Script
  - access via REST
  - provide a simple retrieve mechanism for reading instances and types through REST
  - provide a simple update mechanism for changing instances and types through REST

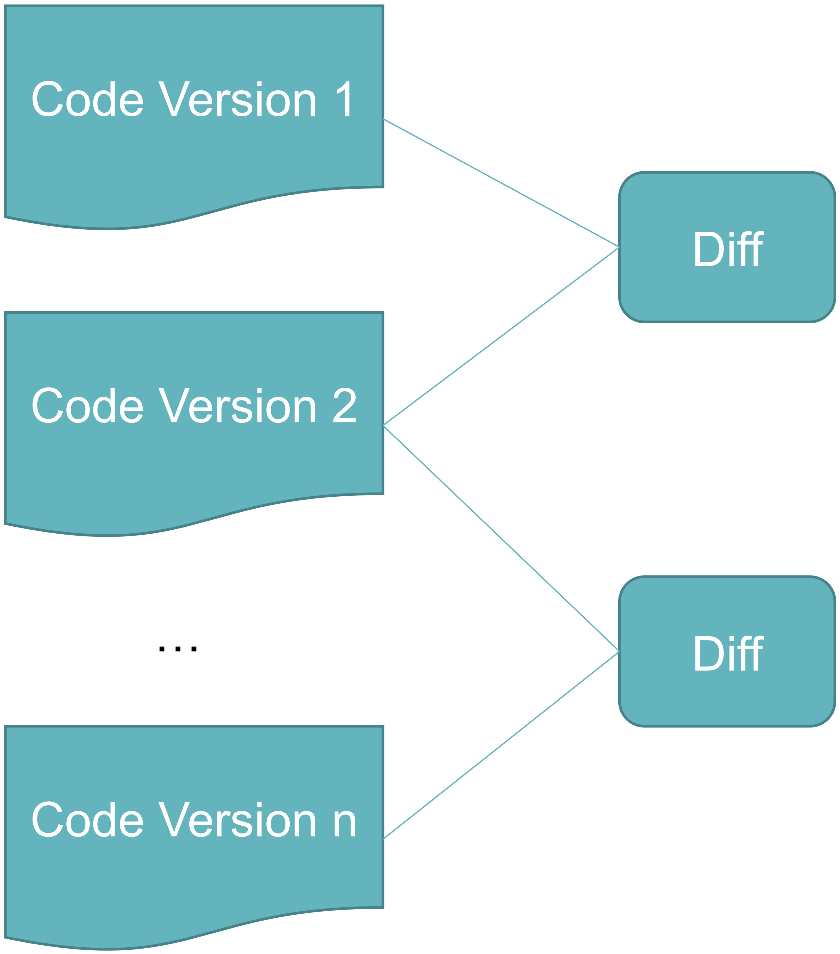
PLANT UML/TEST

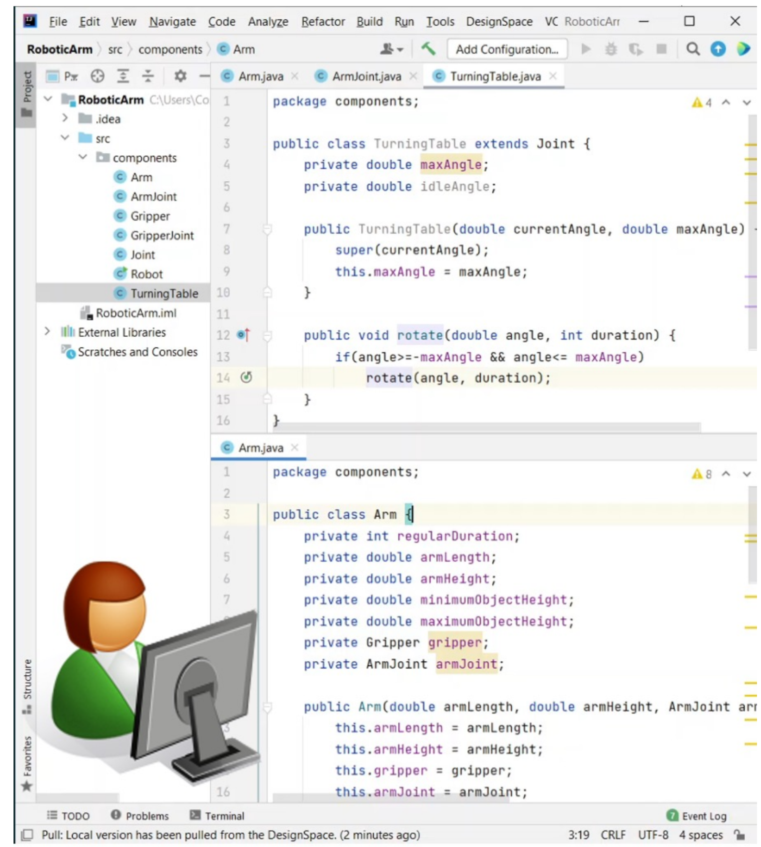
(ALEXANDER.EGYED@JKU.AT)

- http://www.plantuml.com/
- library for visualizing elements, perhaps understanding location of element for visual support

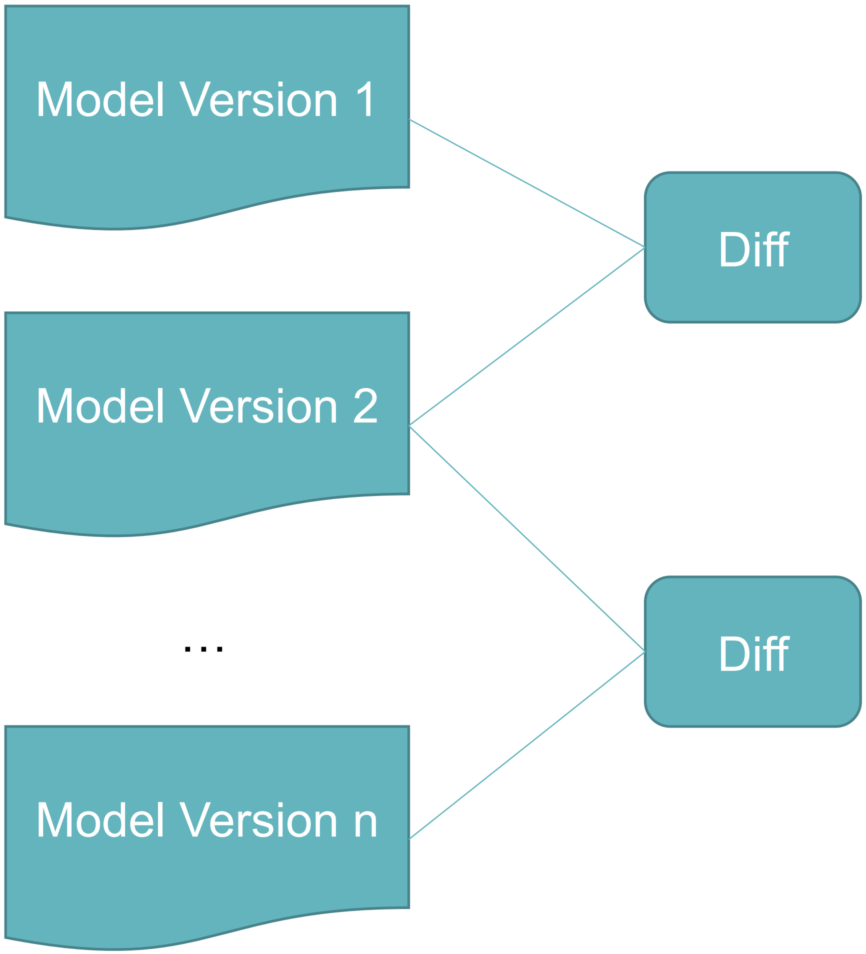


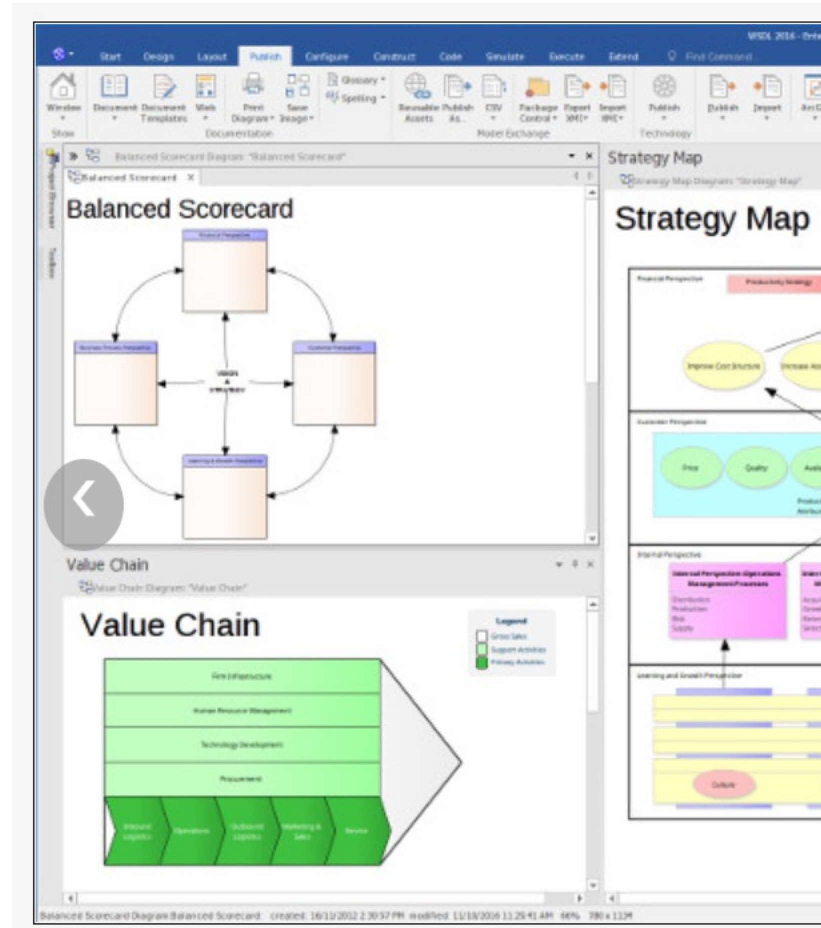
IMPORT CODE THROUGH VERSION HISTORY





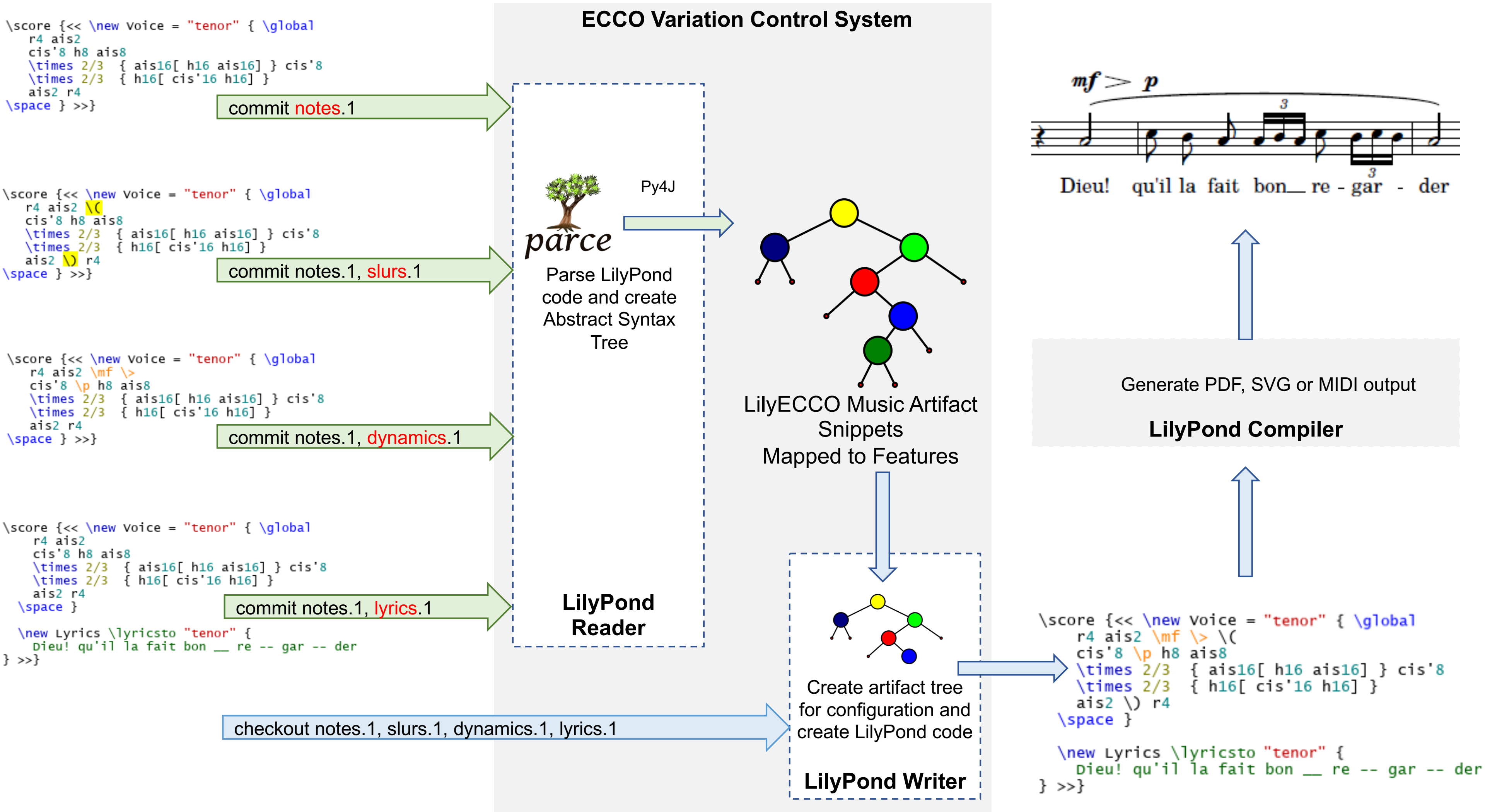
IMPORT MODEL THROUGH VERSION HISTORY/REFACTORING





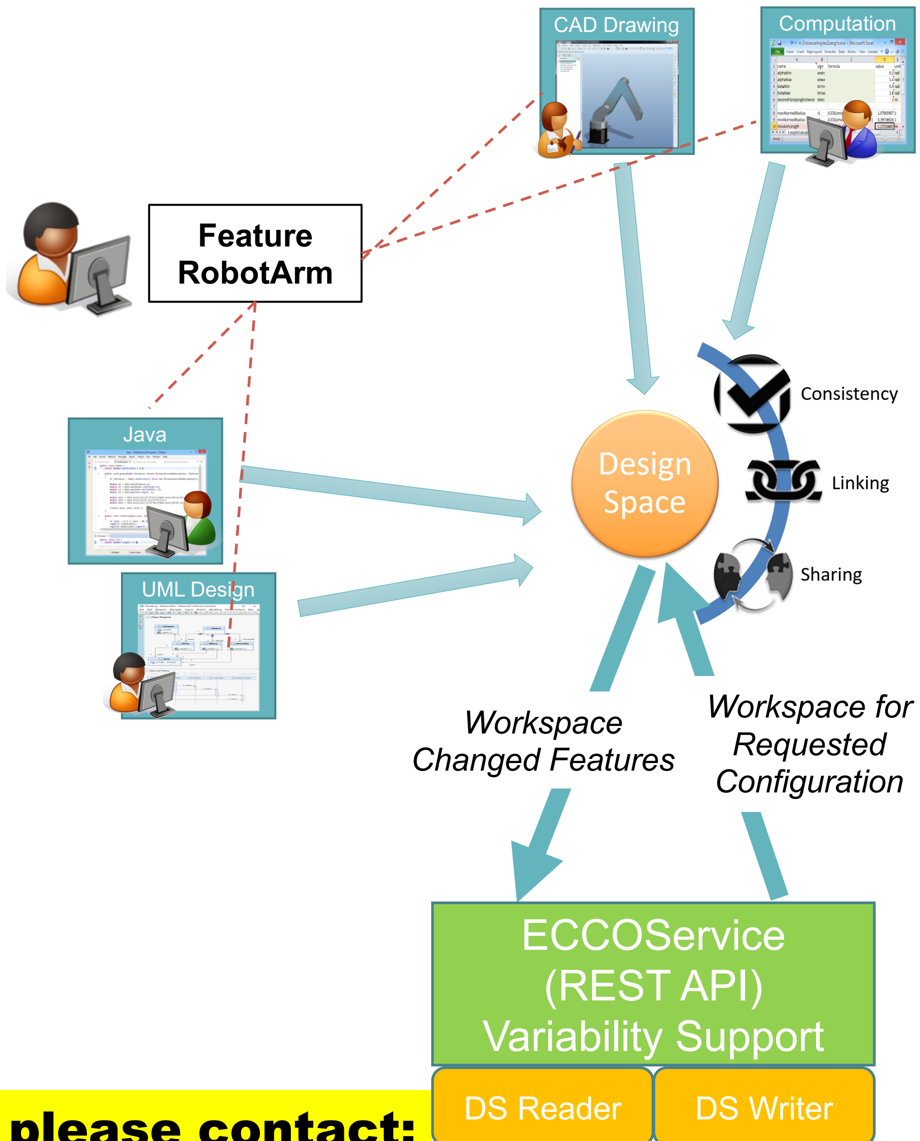
Integration of ECCO and DesignSpace

ECCO VARIATION CONTROL SYSTEM (EXAMPLE: DIGITAL MUSIC PUBLISHING)



INTEGRATING THE DESIGN SPACE WITH ECCO

- Goals
  - Extend the DesignSpace to benefit from ECCO's variability support
  - Reuse plugins developed for DesignSpace in ECCO
- Task
  - ECCO Plugin reading and writing the DesignSpace data structure
  - Demonstration for Java and Visio (existing DesignSpace plugins)

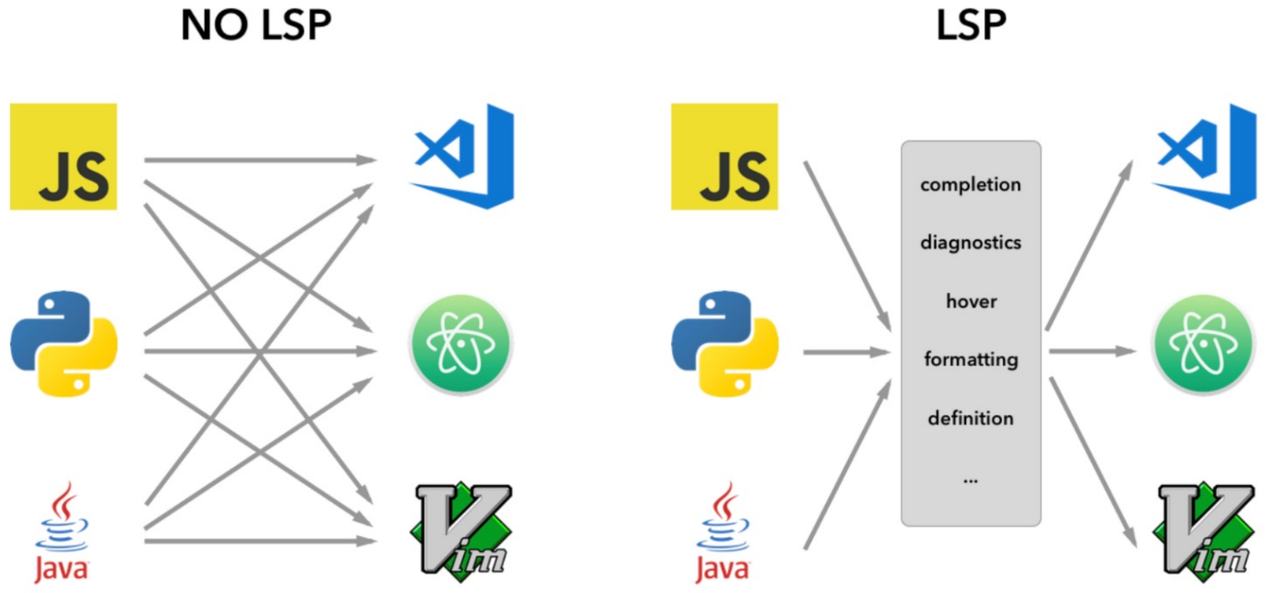


For this topic please contact: paul.gruenbacher@jku.at

JKU

ECCO and the Microsoft Language Server Protocol

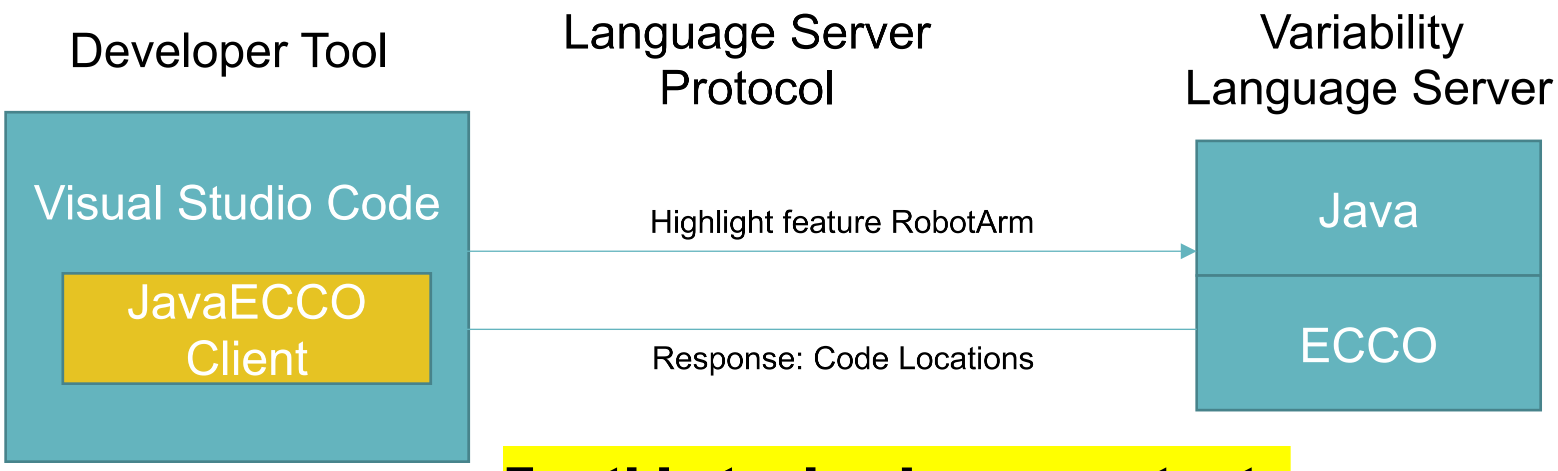
MICROSOFT LANGUAGE SERVER PROTOCOL



- IDE features like auto-completions or Go to Definition requires writing a domain model (a scanner, a parser, a type checker, a builder and more)
- A Language Server provides these features in its own process.
- The language server protocol (LSP) defines the messages exchanged between a development tool and a language server process.

LANGUAGE SERVER WITH VARIABILITY

- Goals
  - Study the LSP and its extensibility features
  - Define protocol extensions for ECCO (e.g., highlight feature in code, hide features, etc.)
  - Use ECCO REST API to implement extensions
  - Test with IDE



For this topic please contact:
paul.gruenbacher@jku.at