3 Grundlagen
TODO

3.1 Die MechatronicUML und ihre Tool-Suite
[Von Proposal]

3.2 Modell-Transformations-Ansatze [Nicht mehr relevant ???]
[Von Proposal]

3.5 MechatronicUML plattformunabhängiges Modellieren [Nicht mehr relevant ???]

3.6 MechatronicUML Hardware-Plattform-Beschreibungs-Modellierung [Nicht mehr relevant ???]

3.7 Modellbasierte Codegeneration [Nicht mehr relevant ???]

3.8 Continuous Integration/Continuous Deployment
[Von Proposal]

3.9 Über die Arduino-CLI:

Die Arduino-CLI ist eine offizielle kommandozeilenbasierte Software für das Arbeiten mit Arduinocode und Boards, auf die mit der Software von Arduino zugegriffen werden kann. Sie wurde nach den Installationsproblemen von „Eclipse C++ Tools for Arduino 2.0“ (siehe 6.2 TODO: Verweis) bei einer Suche nach der Verwendung von Arduinosoftware per Kommandozeile gefunden (Eingabe: „arduino command line“). Sie wurde statt des Erforschens der von der Arduino-IDE verwendeten Befehle gewählt, weil sich ihre Nutzung beim Lesen ihrer offiziellen Dokumentation als leichter und zuverlässiger nutzbar herausgestellt hat.
In vorläufigen Nutzbarkeitstests konnten über die von ihr bereitgestellten Kommandozeilenbefehle erfolgreich alle Schritte, die über die Arduino-IDE erfolgten, durchgeführt werden. [((((Entfernt: Deshalb wurde im Rahmen ein Eclipse-Plug-In konzipiert, das über eine Bibliothek für Kommandozeilenaufrufe auf die Arduino-CLI zugreift.)))]

3.10 Eclipse-Plug-Ins:

Eclipse wurde als modulares Softwaresystem konzipiert [1, https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fguide%2Fintro%2Fpde_overview.htm]. In diesem Abschnitt werden die für diese Ausarbeitung relevanten Aspekte der Entwicklung von Eclipse-Plug-Ins zusammengefasst.
Hier stellen Plug-Ins einzelne modulare Einheiten dar, die jeweils etwas Code gruppieren und ihre Anforderungen wie Java-Packages und andere Plug-Ins sowie die bereitgestellten eigenen Java-Packages und Funktionen für Eclipse spezifizieren [2, https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Fplugin.htm&cp%3D4_1_3]. Solche Informationen werden in den Manifest-Dateien wie plugin.xml und MANIFEST.MF festgehalten [https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Ffirstplugin_create.htm].

Im Sinne der Modularität sollen sie nur eine lose Kupplung aufweisen und dies wird mitunter durch Extension und Extension-Points erreicht [1, https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fguide%2Fintro%2Fpde_overview.htm].

Extensions sind der Ansatz für das Hinzufügen von Verhaltensmöglichkeiten und GUI-Elementen [1, https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fguide%2Fintro%2Fpde_overview.htm] zu Eclipse [https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Ffragment.htm&cp%3D4_1_2]. Ihre Definitionen geben Informationen wie die jeweils zugrunde liegende Klasse, aber auch Bedingungen wie Namensfilter an [https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Ffragment.htm&cp%3D4_1_2]. Export-Wizards werden ebenfalls über Extensions zu Eclipse hinzugefügt [https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fdialogs_wizards_extensions.htm].
Extension-Points werden bei einem Bezug auf Code wie eine angegebene Klasse oder ein Paket verwendet und ihre Möglichkeiten reichen von nutzenden Zugriffen wie Funktionsaufrufen zu dem Hinzufügen oder Ersetzen von Code [https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Ffragment.htm&cp%3D4_1_2].

Die Ausführung einer Eclipse-Installation mit eigenen geschriebenen Plug-Ins kann auf zwei Wegen erfolgen [https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Ffirstplugin_run.htm&cp%3D2_0_2_4]:
1.: Der Build-Vorgang des Plug-In-Projekts wird durchgeführt und danach werden die erstellte .jar-Datei und die zuvor erwähnten Manifest-Dateien in dem Ordner „plugins“ an dem Eclipse-Installationsordner kopiert. Beim nächsten Start der Workbench bzw. Arbeitsumgebung wird es erkannt und geladen.
2.: Durch das Starten des Plug-In-Projekts in der Runtime-Workbench. Dies erfolgt per [Rechtsklicken auf den Plug-In-Projekt-Ordner]/“Run As“/“Eclipse Application“. Im Vorfeld erfolgte Tests haben gezeigt, dass zumindest unter der verwendeten Eclipse-Installation aus [https://www.mechatronicuml.org/download.html, Virtual Box Image bzw. Lubuntu LTS 16.04 + MechatronicUML tool suite 1.0 RC 1 (Full-Eclipse Neon x64 Bundle; Released August 2016)] hierbei automatisch alle offenen Plug-In-Projekte gebaut und geladen werden.

Für das leichtere Verteilen von Plug-Ins ist das Exportieren zu Ordnern und .jar-Dateien möglich [2, https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Fplugin.htm&cp%3D4_1_3].