Git-basiertes Qualitätsmonitoring und von Systems Engineering Modellen

Daniel Lehner[1], Simon Vamberszky[1], Konrad Wieland[2], Daniel Siegl[2].

[1]Johannes Kepler Universität, Institut für Wirtschaftsinformatik – Software Engineering, Altenberger Straße 69, 4040 Linz, Österreich , Vorname.Nachname@jku.at
[2]LieberLieber Software GmbH Handelskai 34/5, 1020 Wien Österreich, vorname.nachnahme@lieberlieber.com

Keywords

MBSE, Qualität, KI, Git

Zusammenfassung: Die Abstraktionsmöglichkeiten von Model-Based Systems Engineering (MBSE) erlauben es, mit der Komplexität moderner Systeme effizient umgehen zu können. Mittlerweile sind die Modelle, die beim MBSE erstellt werden, jedoch meist sehr groß und recht unstrukturiert, da sie von Projekt zu Projekt stetig weiterentwickelt werden. Diese Modelle können dann sehr rasch die notwendige Flexibilität und Anpassbarkeit bestehender Prozessen hemmen. Eine Verbesserung dieser Modelle über die Zeit erfordert jedoch eine Information über die tatsächliche Modellqualität, um Schwachstellen zu identifizieren und auszubessern. Solche Informationen sind aktuell für MBSE-Modelle jedoch noch nicht verfügbar. Deshalb stellen wir in diesem Beitrag eine Architektur vor, welche die Messung und Visualisierung von Modellqualität ermöglicht, und diese Aspekte nahtlos in den MBSE-Entwicklungsprozess integriert.

1 Einleitung

Die Verbindung von physischen und digitalen Komponenten in sogenannte Cyber-Physische Systeme lässt die Komplexität dieser Systeme enorm steigen. Zum Umgang mit Komplexität werden im Bereich Systems Engineering bereits seit vielen Jahren die Abstraktionsmöglichkeiten von Modellen genutzt. Diese Modelle sind über die Jahre so sehr gewachsen, dass eine Anpassung mittlerweile sehr kosten- und zeitintensiv ist. Individualisierte Produktgestaltung (Stichwort Loß-Größe 1) und agile Entwicklung erfordern jedoch eine starke Flexibilität und Anpassbarkeit. Da diese Eigenschaften in vielen MBSE-Modellen nicht mehr gegeben sind, werden etablierte Prozesse oft ersetzt und durch aufwendige Workflows umgangen.
Um also eine nachhaltige Entwicklung von Systems Engineering Modellen und den dahinterliegenden Systemen zu ermöglichen, finden immer mehr Tools und Methoden aus dem Bereich des Software Engineering Einzug in die Systems Engineering Domäne. Ein prominentes Beispiel dafür ist Git. Aktuell gibt es jedoch noch keine Möglichkeit, bei solchen Workflows, (1) die Qualität von Modellen messbar zu machen, (2) diese zu visualisieren, um Probleme zu identifizieren und Gegenmaßnahmen einzuleiten.

Im Rahmen dieser Arbeit präsentieren wir deshalb einen Ansatz zur Definition, Berechnung sowie Visualisierung beliebiger Modell-Metriken, und demonstrieren diesen Ansatz anhand konkreter Metrik-Beispiele. Das vorgestellte Framework verwendet außerdem Git zum Management von Systems Engineering Modellen und Metriken, um eine nahtlose Integration in bestehende Modellierungsprozesse zu ermöglichen.

2 Git-basiertes, kontinuierliches Qualitätsmonitoring von Systems Engineering Modellen

Zur Realisierung der Berechnung und Visualisierung verschiedener Qualitätsmetriken wird in dieser Arbeit das Git-basierte, kontinuierliche Qualitätsmonitoring von Systems Engineering Modellen vorgestellt.

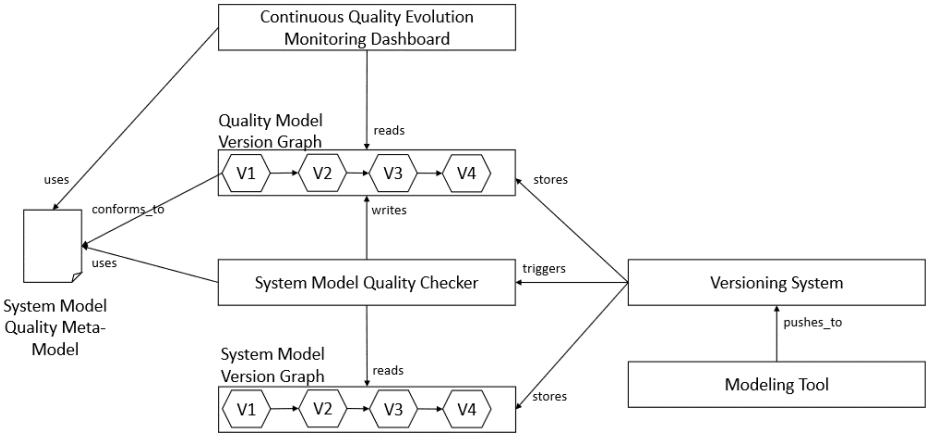

Bild 1: Architektur des vorgestellten Git-basierten Qualitätsmonitoring Ansatzes

Bild 1 zeigt die Architektur dieses Ansatzes. Dabei werden MBSE Modelle in einem Modellierungstool (z.B. Enterprise Architect oder Modelio) entwickelt. Das Versionierungssystem erkennt Änderungen im Modell, und triggert dadurch eine bestimmte Aktion (siehe z.B. Github Actions [1]). Diese Aktion führt den System Model Quality Checker aus. Dieser berechnet auf Basis des System Model Quality Meta-Model, bzw. den für das konkrete Modellierungsprojekt hinterlegten Metriken, das Quality Model der neuen Modellversion. Das System Model Quality Meta-Model (siehe Bild 2) ist ein konzeptionelles Meta-Modell, um verschiedene Qualitätsmetriken beschreiben zu können. Ein Set an Qualitätsmetriken für ein konkretes Modell wird in einem JSON-Dokument serialisiert (siehe Beispiel in Bild 3), und gemeinsam mit dem Systems Engineering Modell im Versionierungssystem abgelegt.

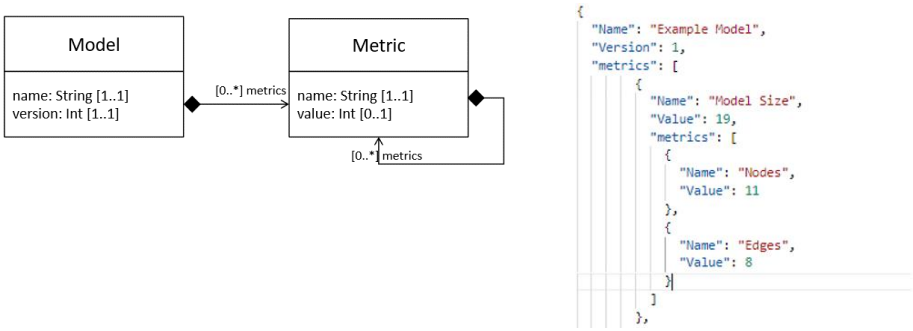

Bild 2: System Model Quality Meta-Model (links), und Ausschnitt eines Beispiel-Qualitätsmodells mit verschiedenen Metriken für ein konkretes Modell, basierend auf dem System Model Quality Meta-Model, serialisiert als JSON.

Das Continuous Quality Evolution Monitoring Dashboard nimmt nun die im Versionierungssystem abgelegten Quality Models für verschiedene Modellversionen, und bereitet die darin gespeicherten Qualitäts-Informationen visuell auf. Dabei gibt es verschiedene Visualisierungen (i) zur Darstellung der Qualität einer konkreten Modellversion, sowie (ii) der Darstellung der Entwicklung verschiedener Qualitätsmetriken über mehrere Modellversionen hinweg (siehe Bild 3).

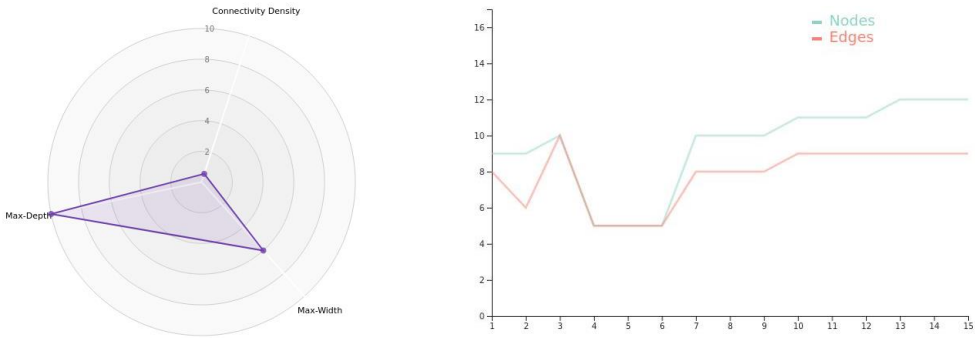

Bild 3: links: Spider Diagramm einer bestimmten Version eines konkreten Modells, für die Metriken Connectivity Density (Anzahl Elemente / Anzahl Verknüpfungen zwischen den Elementen), Max-Depth (maximale Verknüpfungstiefe) , und Max-Width (maximale Anzahl an Verknüpfungen in einer Ebene). Rechts: Liniendiagramm zur Visualisierung der Werte zweier Metriken (#Knoten und #Kanten im Modell) über die verschiedenen Modellversionen (1-15) hinweg.

2.1 Integration mit Git

Das vorgestellte Framework verwendet dabei das bereits etablierte Versionierungssystem Git, um (i) Informationen abzulegen, bzw. (ii) den gesamten

Monitoring Prozess automatisiert zu triggern. Da Git bereits automatisiert von Modellierungstools wie Enterprise Architekt verwendet werden kann, ermöglicht dies nun eine nahtlose Integration des vorgestellten System Model Quality Checking mit existierenden Modellierungsprozessen. Aus User-Sicht bedeutet das, dass nach jedem Checkin (dieser kann z.B. durch Speichern des Modells getriggert werden) vollautomatisiert sämtliche definierten Metriken berechnet werden, und der aktuelle Qualitätsstand direkt nach der Berechnung im System Model Quality Dashboard vorhanden ist. Falls nun auf Basis einer Visualisierung ein Problem mit der Modellqualität festgestellt wird, kann dieses ganz einfach durch entsprechende Änderungen im Modellierungstool behoben werden. Die Auswirkung dieser Änderung auf die Modellqualität ist dann nach erneutem Checkin umgehend im System Model Quality Dashboard sichtbar.

3 Metriken

Der vorgestellte Ansatz ist durch das generische System Model Quality Meta-Modell auf beliebige Metriken anwendbar. Dieses Kapitel zeigt, wie nun konkrete Metriken definiert werden können, um das vorgestellte Monitoring zu ermöglichen. Um eine neue Metrik zu definieren, müssen konkret folgende Informationen festgelegt werden:

- **Metrik-Name:** wird verwendet, um Bezug auf die Metrik zu nehmen (innerhalb eines Quality Models, oder bei den Visualisierungen des Continuous Quality Evolution Monitoring Dashboards).

- **Berechnungsfunktion**: wird im System Model Quality Checker hinterlegt, und gibt an, wie aus den Modellen konkrete Metrik-Werte berechnet werden. Zum Abrufen von nötigen Modellinformationen werden dafür Modell-Queries verwendet. Für solche Queries gibt es bereits eigene Sprachen, wie z.B. OCL [4].

Anschließend wird die Metrik nach jedem Checkin automtisch vom System Model Quality Checker berechnet, und der ermittelte Wert kann mittels des Quality Model Version Graphs vom Continuous Quality Evolution Monitoring Dashboard dargestellt werden.
Neben Metriken, die direkt auf Basis von Modellinformationen berechnet werden, unterstützt das System Model Quality Meta-Modell auch aggregierte Metriken. Diese Metriken errechnen sich aus verschiedenen Sub-Metriken, welche in weiterer Folge als Teil der Metrik definiert werden müssen.

3.1 Ein Beispiel

Im Folgenden veranschaulichen wir diese Metrik-Definition anhand der Metrik „Model Size“ aus Bild 2. Diese Metrik gibt einen Überblick über die Größe eines Modells, indem die einzelnen Modellelemente durch Aggregierung verschiedener Sub-Metriken gezählt werden. Um dabei möglichst unabhängig von konkreten Modellen zu sein, wird für diese Beispiel-Metriken auf Grafen als Grundlagen jeglicher Modelle

zurückgegriffen. Im Grunde kann jedes Modell also als Graf gesehen werden, der aus Knoten und Kanten besteht. Diese Knoten und Kanten können jeweils gezählt werden (in Bild 2 dargestellt über die Metriken Nodes bzw. Edges). Die Aggregierungsfunktion, um von diesen beiden Metriken nun zur Metrik „Model Size“ zu kommen, ist dabei relativ einfach: Nodes + Edges. Bild 3 zeigt nun die Visualisierung der für konkrete Modellversionen ermittelten Metriken Nodes und Edges über die Zeit als Linien-Diagramm.

4 Zusammenfassung und Ausblick

Wir haben gezeigt, dass ein stetiges Qualitätsmonitoring auf Basis etablierter Prozesse wie Git nicht nur möglich ist, sondern einen erheblichen Mehrwert in der Modellierung von Systemen bringen kann, um die immer komplexeren Modelle auf einem qualitativ hochwertigen Niveau weiter entwickeln zu können.
Der vorgestellte Ansatz erlaubt dabei die automatisierte Ermittlung, Speicherung und Visualisierung beliebiger Metriken. Wir haben dies anhand dreier Beispiel-Metriken veranschaulicht, die für jede beliebige Modell-Art anwendbar sind. Diese Basis-Metriken können in konkreten Projekten auf Basis der in Kapitel 3 beschriebenen Methodik für konkrete Modellarten (bei SysML BDDs können z.B. Knoten in Blöcke, Properties, Aktuatoren, etc. unterschieden werden) oder Anwendungsfälle (z.B. Untersuchung der Paketierung von Modellen anhand von Kohäsion und Kopplung) erweitert werden. In Zukunft kann die Entwicklung und Evaluierung eines konkreten, etablierten Sets an Metriken auf Basis des in diesem Papier vorgestellten Ansatzes dabei helfen, ein besseres Verständnis von Modellqualität in verschiedenen Domänen zu bekommen. Zusätzlich können durch verschiedene KI-basierte Ansätze Möglichkeiten zur automatisierten Optimierung von Modellen auf Basis errechneter Metrik-Werte gefunden werden.

Referenzen

[1] Kinsman, T., Wessel, M., Gerosa, M. A., & Treude, C. (2021, May). How do software developers use GitHub Actions to automate their workflows?. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (pp. 420-431). IEEE.

[2] Mirjalili, S. (2019). Genetic algorithm. In Evolutionary algorithms and neural networks (pp. 43-55). Springer, Cham.

[3] M. Fleck, J. Troya, M. Wimmer: Towards Generic Modularization Transformations. (2016). International Workshop Modularity in Modelling (MOMO 2016), in conjunction with MODULARITY, Malaga, Spain; 15.03.2016, in Proceedings of the International Workshop Modularity in Modelling. ACM.

[4] Richters, M., & Gogolla, M. (2002). OCL: Syntax, semantics, and tools, in Object Modeling with the OCL (pp. 42-68). Springer, Berlin, Heidelberg.