Journal of Visual Language and Computing

journal homepage:

Auto-Modularity Enforcement Framework Using Micro-service Architecture

Hanzhong Zheng, Justin Kramer and Shi-Kuo Chang*

Department of Computer Science, University of Pittsburgh, 6135 Sennott Square, 210 S Bouquet St., Pittsburgh, PA, USA, 15260-9161

ARTICLE INFO

Article History:
Submitted 10.22.2020
Revised 11.15.2020
Second Revision 12.2.2020
Accepted 12.10.2020

Keywords:
Micro-service
Automatic software development
Service-oriented architecture
Modularity enforcement
Visual software development

ABSTRACT

The evolution of the software architecture has been progressively shifting to emphasize modularity, isolation, scalability, agility, and loose coupling. Service-oriented architecture (SOA) has started to gain popularity in this direction. Micro-services are a lightweight SOA that aim to largely scale applications while ensuring isolation and distribution. Modularity is sometimes left behind or difficult to achieve with fine-grained distribution of programmer responsibilities. In this paper, we propose an automatic modularity enforcement (AME) framework during the software development life cycle (SDLC) through intermediate representation. Our idea was inspired by automatic software development for building a scalable application. We implemented this framework to support visual software development using the Java Spring Boot Micro-service tool.

1. Introduction

Software architecture reflects the definition of all interacting components in the system for satisfying customers' requirements. Nowadays, analytic applications largely increase the criticality of the software quality and scalability. In the micro-service paradigm, scalability and isolation are improved through dividing a large application service into several sub-services, which are independently deployed and communicated through interfaces via standard data formats and protocols such as XML, HTTP, etc. [1]. Each sub-service implements the partial functionality of the entire system.

The majority of a software system is divided into several modules during the design. However, modularization has always been one of the greatest challenges in software architecture design. Enforcing modularization can also be considered as an NP-hard problem for programmers [2]. Modularization separates the program's functionality into several modules. Each module is independent and interacts with each other. Inadequate modularization can easily influence the distribution, persistence, isolation, and even the overall software quality. For programmers, modularization is a key but challenging principle. The complexity of modern software systems makes them much harder for programmers to understand and maintain, especially with respect to scalability. Modularization can allow for decomposition of the software system to reduce the complexity and improve the maintainability. Furthermore, modularity can make the application more tolerant of uncertainty.

To ensure the continuous delivery of trustworthy and high-quality software systems while reducing the burdens on programmers, automation in software development has become important. Many efforts have been made in recent years in automation of the software development process under three categories: Rapid Application Development (RAD) [3] [4], Code generation [5], and Model-Driven Architecture (MDA) [6] [7]. In object-oriented programming, modularity and encapsulation are closely tied to each other and play dominant roles. Enforcing modularity can limit the propagation of program errors and establish software maintainability. The mechanism of automatic module enforcement (AME) allows the program modules to be developed in a customized and organized way. AME sets more constraints in order to keep consistency and cohesion in the entire software system design. In this

*Corresponding author
Email address: schang@pitt.edu
ORCID: 0000-0003-0426-4030

DOI reference number: 10.18293/JVLC2020N2-008

paper, we propose an automatic module enforcement framework that automatically generates and enforces the modularity in software development from the software system design. This framework is flexible and agile for adapting into other programming languages. The contributions of this paper are as follows:

1. We developed a new time-critical application design system that specify different interaction patterns among components in the software system design.

2. We proposed an automatic modularity enforcement (AME) framework using the concept of micro-service architecture to generate and reinforce the module's functionality and cohesion.

3. We implemented our framework on a well-defined experimental system using the Java Spring Boot developing template.

2. Related Work

2.1 Service-Oriented Architecture (SOA)

Enterprises have increased their demand for flexible, efficient and extendable architecture paradigms in the current highly competitive software market. SOA is a service-based architectural style that usually is viewed as a black box that may have many underlying services [8], but brings many significant benefits to Enterprises in the way of flexibility, agility and high degree of collaboration between business and IT. The flexibility of SOA demonstrates how legacy applications can easily integrate with new applications. SOA has the ability to quickly respond to ever-changing requirements and demands. The main goal of SOA is to support a business process that reflects their collaboration.

The communications in a SOA commonly utilize WSDL (Web Service Description Language), UDDI (Universal Description Discovery and Integration), and SOAP (Simple Object Access Protocol) among Service

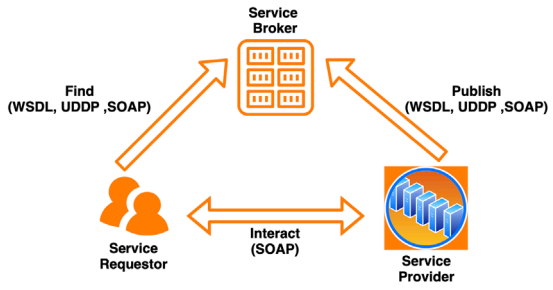

Figure 1: Communication structure of SOA

Provider, Service Broker and Service Requestor/Consumer (Fig. 1). Service Provider offers a variety of different services that are ready to use. Service Requestor demands the services. Service Broker is a service registry for connecting the Service Provider and Service Requestor.

The limitations of the SOA are also obvious. The communications between services mainly depend on message passing, which can easily become overwhelming when applications require heavy data exchange. The connections are exponentially increasing for a server due to transmission protocols, and SOA is costly in deployment and human resources.

2.2 Monolithic vs. Micro-service Architecture

The waterfall development model and associated technology are representations of traditional software development processes, which usually require a large team on a monolithic artifact. In the monolithic architecture, the main concept is "single": A monolithic application is built from a single unit, which is self-contained and independent from other applications. However, a great service design for a large application should be stateless and allow the application to scale vertically. Micro-services arrange an application to be a collection of loosely coupled, interconnected modular services where individual services communicate through REST APIs and lightweight messages. To achieve this isolation each service should be independent from other services. Fig. 2 illustrates an example of a micro-service architecture.

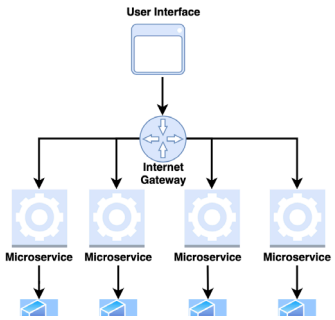

Figure 2: An example of a micro-service architecture

Maintainability, scalability and reliability are the main drawbacks of the monolithic architecture, and issues concerning them are proliferated in the current enterprise market. Micro-services ensure the continuous delivery and deployment of a large and complex application associated with scalability, testability, flexibility and fault tolerance. Service failure is unpredictable but harmful. Isolation in micro-services ensure the application continue to operate even if there is a service failure. Enterprise applications have the essence to be complex and highly demanding of

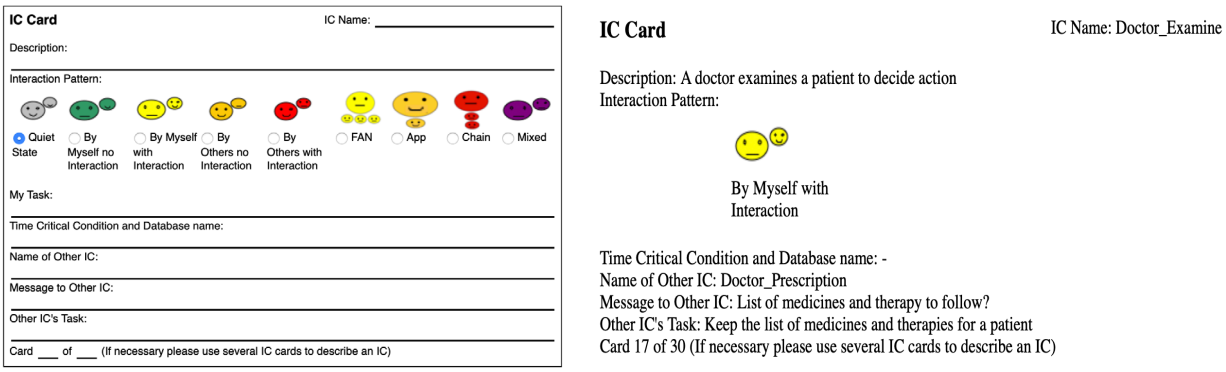

Figure 3: An IC card example of creating a functionality component for 'Doctor_Examine'

scalability and responsiveness. The benefits of micro-services seem to fulfill those requirements and attract business Enterprises transition from monolithic to micro-service architecture. As one of the biggest e-commerce company in Europe, Otto Group started to build their system using micro-services, which vertically decomposes their system into four loosely coupled applications: Product, Order, Promotion and Search/Navigation [9]. Another example is the Netflix. Netflix is the top Internet television network in the world and spent over 7 years on the transition to the micro-service. The successful transition allows the video to be displayed on a variety of screen sizes and platforms [10]. Besides, micro-services can easily integrate with popular cloud platforms. Amazon web services and Microsoft Azure both deploy micro-services on their cloud platforms.

2.3 Software Development Automation (SDA)

Software development in general requires large amounts of human endeavor. The improvement of computer architectures and networks drives the size of computer software and their diverse platforms [13] exponentially, increasing in order to satisfy the increasing needs of providing more software features. Automation in software development refers to replacing repeatable processes and reducing manual intervention, which accelerates the delivery of high-quality software products [11]. Automation in the software space focuses on building both software and testing automation, which usually takes a significant amount of time. Application building involves many steps, like code updating, compiling, and deployment. Software testing intends to discover bugs, errors, or defects during the execution of programs or application.

Software projects range from small scale personal projects to large scale industrial applications. This triggers the popularity of open software repositories such as Github, Sourceforge, and Bitbucket. However, well-maintained software development frameworks, like Sot, Wala, LLVM, all require a successful build process of the project repositories. Foyzul Hassan et al. present a feasible automatic software binding on Java projects on the state-of-the-art version control repository [12]. They found that 57% of build failures can automatically be resolved. Software testing is an intensive and costly task in the software development life cycle (SDLC). Automated software testing aims to reduce the workload through automated testing. Test automation largely impacts the quality, development time, and cost of the software to the market [16]. Automated software testing can be unit testing, functional testing, testing management tools, and so on.

Modularity is an important concept in software applications. It enhances the reusability of the previous code. Modules usually are divided based on their functionality, but they work together for serving a specified business domain. Modularization is always the main issue in SDLC and the core task for programmers [17] [18] [19]. One of the benefits of micro-services is the enhancement of modularity in the project to achieve fine-grained distribution of sub-services. For object-oriented programming, modularization is necessary for development teams. The high benefits of modularity certainly associate with the challenges in software design and implementation. We propose an automatic modularity enforcement framework from the software design to the software implementation process. The implementation of our framework utilizes micro-services to enforce isolation and reusability. We also demonstrate that flexibility by not only automatically creating Java modules, but also by automatically creating modules in other object-oriented programming languages. The IC card can model the interaction patterns for designers to choose, illustrated in different colors and emoticons with associated names. Interaction patterns define how statuses are communicated with other IC cards.

3. Time Critical Condition Design

Our IC card management system (ICMS) is used for designing time critical applications. An IC card is a visual specification scheme for rapidly prototyping the entities of an application [20]. The ICMS is a visual tool that allows the creation, edition, visualization, and exporting of one or more IC cards. The connection of

the ICMS and our auto-modularity enforcer utilizes XML specification. The ICMS automatically generates XML specifications. Fig. 3 is an example of the IC card example that can show the structure. There are 8 interaction patterns: 'quiet state', 'By myself no interaction', 'by Myself with interaction', 'By Others no interaction', 'By Others with interaction', 'FAN' (fan-out), 'App', 'Chain', and 'Mixed'. The 'Quiet' state indicates not working or in a restful state.

'FAN' indicates the distributed fan-out of a larger task to a number of smaller tasks. 'My task' is the task assigned to this IC card. The content of the IC card provides the detailed descriptions of the task. For example for a brainwave sensor component IC card, "(1) if T_c > threshold T; calculate two options states: attention and mediation states; otherwise, keep collecting the EEG. (2) if medication value > attention value, send the medication state and value to the database server; else: send the attention state and value to the database server." 'Name of Other IC' specifies the other IC cards that interact with each other. 'Messages to Other IC' contains the message format and message content (e.g. "msg1: raw EEG data, msg2: user state, value"). After finishing filling out the fields of the IC card, the designer submits the IC card to the database, and the IC card database automatically generates the other designated number of IC cards with temporary information so that the designer can edit them at any time. In addition, the XML schema has also been automatically generated and can be downloaded for the next component to transform into java modules.

4. Experimental Tool

For our experimental tool, we began with an analysis of the micro-service architecture. The micro-service paradigm focuses upon modularity in backend development by introducing componentization of the services it defines. These services are broadly defined within the framework, and the architecture behind these services varies vastly from one organization to another. With the growth of the micro-service architecture in recent years [9], the need for Rapid Application Development (RAD) [3] [4] in the space has expanded. Based upon this analysis, we developed a tool to enhance the reusability, cohesion, and distribution of micro-service development while reducing coupling. Our experimental tool is driven by an automatic modularity enforcement framework based upon the Java Spring Boot framework. Java Spring Boot supports the creation of a framework that utilizes fine-grained distribution of sub-services while allowing for vast extensibility through Cloud, API, and serverless interfaces to services.

We employed Python to engineer a dynamic auto-generation tool within our Java Spring Boot micro-service architecture. The Python program allows for flexible micro-service generation based upon a standard XML input interface, connecting to our IC card management system that produces the XML files. With our Python program, users can specify constant factors to identify their central repository, the XML file to target, and the title of their template files. To allow for extensibility, the Python Auto-Generator provides a basic method-layer API to process user-defined micro-services which fit into the templated IC card-based XML structure. This approach allows users to create micro-services through the framework of their choice if basic constraints on input and output are met.

Through the development of the experimental micro-service Auto-Generator (Fig. 4), it is possible to automatically generate micro-service components within a structure that lends itself to extensibility, tight cohesion, loose coupling, and modularity. The XML specification, based on the IC card template, serves as an interface to the Auto-Generator. An XML-based communication structure informs the Auto-Generator which micro-services to create, as well as the database tables to establish for each micro-service. Also, the Auto-Generator establishes micro-service file components in a directory structure specified in the source project directory.

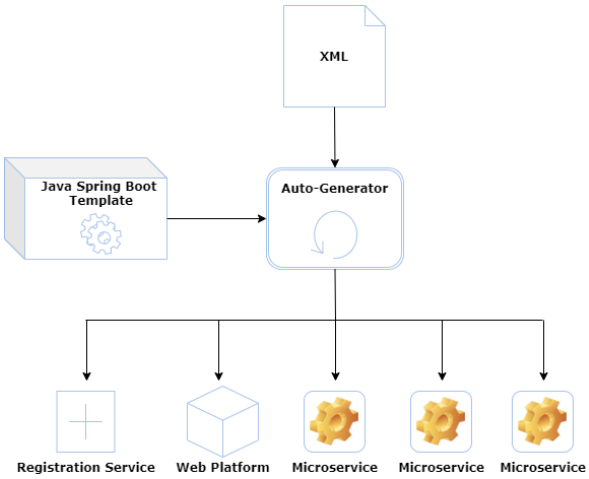

Figure 4: modularity auto-enforcement framework using micro-service architecture

As an example, the *fan* software structure is illustrated in Figure 5. This is for the *get user location* module of an app. At the top we have *track_tracks_service* and under that, as shown in Figure 6, we have two IC cards, *close_closes*, and *distance_distances*. The module (IC card) *track_tracks_service* has a database that stores the location, the latitude, and the longitude. The IC cards are shown in Figure 6. Once the IC cards are defined, the ICMS can output an XML specification as shown in

Figure 7.

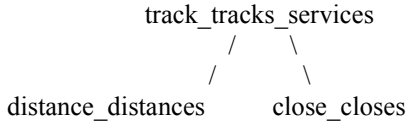

Figure 5. Fan software structure.

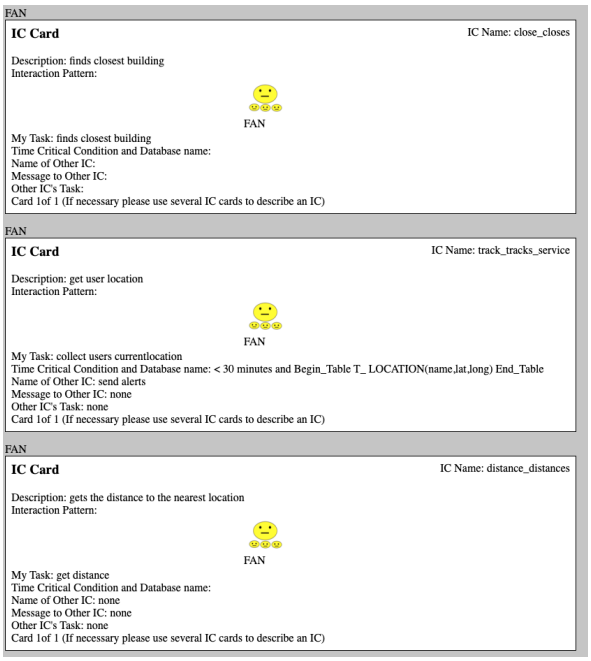

Figure 6. The IC cards for the *fan* software structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<icCardList xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <icCardEntry icEntryId="2804" icEntryName="test">
      <icCard icId="10279" icName="close_closes"
icDescription="finds closest building" icIntPattern="FAN"
icMyTask="finds closest
       building" icTimeCriticalCondition="" icNumberCurrent="1"
icNumberTotal="1">
         <icOther icOtherName="" icOtherMessage=""
icOtherTask="" otherId="-1" />
      </icCard>
      <icCard icId="10277" icName="track_tracks_service"
icDescription="get user location" icIntPattern="FAN"
icMyTask="collect users
      currentlocation" icTimeCriticalCondition="&lt; 30 minutes and
Begin_Table T_ LOCATION(name,lat,longatiude) End_Table"
      icNumberCurrent="1" icNumberTotal="1">
         <icOther icOtherName="send alerts" icOtherMessage="none
" icOtherTask="none " otherId="4784" />
      </icCard>
      <icCard icId="10278" icName="distance_distances"
icDescription="gets the distance to the nearest location"
icIntPattern="FAN"
         icMyTask="get distance" icTimeCriticalCondition=""
icNumberCurrent="1" icNumberTotal="1">
         <icOther icOtherName="none" icOtherMessage="none"
icOtherTask="none" otherId="-1" />
      </icCard>
   </icCardEntry>
</icCardList>
```

Figure 7. The XML specfication.

Based upon the XML specification of the IC cards, the AutoGenerator can then create the output modules. A portion of a module is shown in Figure 8.

```
package io.pivotal.Micro-services.patients;

import java.io.Serializable;
import java.math.BigDecimal;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 * Persistent patient entity with JPA markup. Patients are stored in
an H2
 * relational database.
 *
 * @author Paul Chapman
 */
@Entity
@Table(name = "T_PATIENT")
public class Patient implements Serializable {

        public static Long nextId = 0L;

        @Id
        protected Long id;

        protected String number;

        protected String name;

        protected String address;

        /**
         * This is a very simple, and non-scalable solution to
generating unique
         * ids. Not recommended for a real application. Consider
using the
         * <tt>@GeneratedValue</tt> annotation and a sequence
to generate ids.
         *
         * @return The next available id.
         */
        protected static Long getNextId() {
                synchronized (nextId) {
                        return nextId++;
                }
        }

        /**
         * Default constructor for JPA only.
         */
        protected Patient() {
        }
```

Figure 8. A generated module.

The outputted modules, as demonstrated by the example module in Figure 8, provide two key functions. The first function is a set of class variables that define the database table for the service. The AutoGenerator establishes a table for the micro-service in the database for usage based on the template database. Furthermore, each module provides a method-level API that

controller classes utilize to manipulate and combine the data of each service. The core class of the module is extensible, allowing for the development of ICs within each core class. The ICs inside of each core class define the service's internal communication, logic, and functions. For example, an IC defined as RegisterUser may exist within the User Service. The User Service core class would contain the logic for registering a user, including CRUD (create, read, update, delete) calls to the User database. In addition, a Login Controller may act as a wrapper class around both the User Service and a hypothetical Authentication Service, which it utilizes to authenticate a potential user's information before registration. Lastly, the outputted module communicates with an automatically generated registration server to establish itself as a distributed micro-service. Connections to the registration server are based upon a centralized location that is automatically provided within the creation of each new micro-service.

5. Conclusion

This paper proposes the organization of the generated micro-service, with clear distinctions between Software Quality Assurance (SQA) testing, database creation, data seeding, registration, a web platform, and service methods. The combination of these resources is accessed by Java Spring Boot to compile the micro-service-based software. Based upon our experimental design, micro-services may be created in conjunction with the tenets of Rapid Application Development (RAD) [3] [4]. These micro-services register with a central server, utilize their independent databases, and provide APIs to controllers in the overlying software. Each micro-service interacts with the registration and web components through structured channels based upon naming schemas. Thus the experimental tool provides a basis for our studies pertaining to auto-modularity enforcement framework for micro-services.

Our next research goal is to investigate the optimal organization of the generated micro-services according to some objective functions to minimize, for instance, the total development efforts.

References

[1] Dragoni, N., Lanese, I., Larsen, S., Mazzara, M., Mustafin, R., Safina,L., 2017. Microservices: How to make your application scale.

[2] Prajapati, A., Chhabra, J., 2018. Optimizing software modularitywith minimum possible variations. Journal of Intelligent Systems 29.

[3] Beynon-Davies, P., Carne, C., Mackay, H., Tudhope, D., 1999. Rapidapplication development (rad): An empirical review. Eur. J. Inf. Syst. 8, 211–223.

[4] Berger, H., Beynon-Davies, P., Cleary, P., 2004. The utility of a rapidapplication development (RAD) approach for a large complex information systems development., 220–227.

[5] Liao, H., Jiang, J., Zhang, Y., 2010. A study of automatic code generation, in: 2010 International Conference on Computational and Information Sciences, 689–691. doi:10.1109/ICCIS/2010.171.

[6] Newman, M.E.J., Girvan, M., 2004. Finding and evaluating community structure in networks. Phys. Rev. E. 69, 026113.

[7] Pastor, O., Molina, J.C., 2007. Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer-Verlag, Berlin, Heidelberg.

[8] Cloutier, Robert. 2008. Model Driven Architecture for Systems Engineering. Presentation Slides), Stevens Institute of Technology, presented at INCOSE International Workshop.

[9] Chapter 1: Service Oriented Architecture (SOA). *msdn.microsoft.com*. Archived from the original on February 6, 2016. Retrieved September 21, 2016.

[10] Hasselbring, W., Steinacker, G., 2017. Microservice architectures forscalability, agility and reliability in e-commerce, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 243–246.

[11] Vučković, J., 2020. You Are Not Netflix. Springer International Publishing, Cham. 333–346.

[12] Ohno, O., Furuhata, Y., Komuro, H., Imajo, T. and Komiya, S. 2002. Automated software development based on composition of categorized reusable components—construction and sufficiency of skeletons for batch programs. Electron. Comm. Jpn. Pt. II, 85: 50-66.

[13] Hassan, F., Mostafa, S., Lam, E.S.L., Wang, X., 2017. Automaticbuilding of java projects in software repositories: A study on feasibility and challenges, in: 2017 ACM/IEEE International Symposiumon Empirical Software Engineering and Measurement (ESEM), .38–47.

[14] Moran, K., 2018. Automating software development for mobile computing platforms (doctoral symposium). ArXiv abs/1807.07171

[15] Wedikian, Z., Ayari, K., Antoniol, G., 2009. Mc/dc automatic testinput data generation, in: Proceedings of the 11th Annual Conferenceon Genetic and Evolutionary Computation, Association for Comput-ing Machinery, New York, NY, USA. 1657–1664.

[16] Kumar, D., Mishra, K., 2016. The impacts of test automation on software's cost, quality and time to market. Procedia Computer Science79, 8–15.

[17] Garousi, V., Elberzhager, F., 2017. Test automation: Not just for testexecution. IEEE Software 34, 90–96. doi:10.1109/MS.2017.34

[18] Serme, G., 2013. Modularization of security software engineering indistributed systems. (modularisation de la sécurité informatique dansles systèmes distribués).

[19] Mitchell, B.S., Mancoridis, S., 2006. On the automatic modularization of software systems using the bunch tool. IEEE Trans.Softw. Eng. 32, 193–208.

[20] Hare, E., Kaplan, A., 2017. Designing modular software: A casestudy in introductory statistics. Journal of Computational and Graphical Statistics 26.

[21] Chang, S. K., Rajnovic, P., Zalar, M., 2007. IC Card: Visual specification for rapid prototyping of time-critical applications. International Journal of Software Engineering and Knowledge Engineering 17, 557–573.