ASSIGNMENT #5

Cason Konzer 8/7-9/21

UNIVERSITY OF MICHIGAN

Flint Campus

[CSC-382]: Software Engineering

Contents

Write a three-page paper:

i)Choose one of the two papers listed above.

ii)Summarizes the content of the paper. (minimum 1 page)

iii)Your opinion on how the topic of this paper could be of consequence to a new Software Engineer (potentially you soon). (minimum 2 pages)

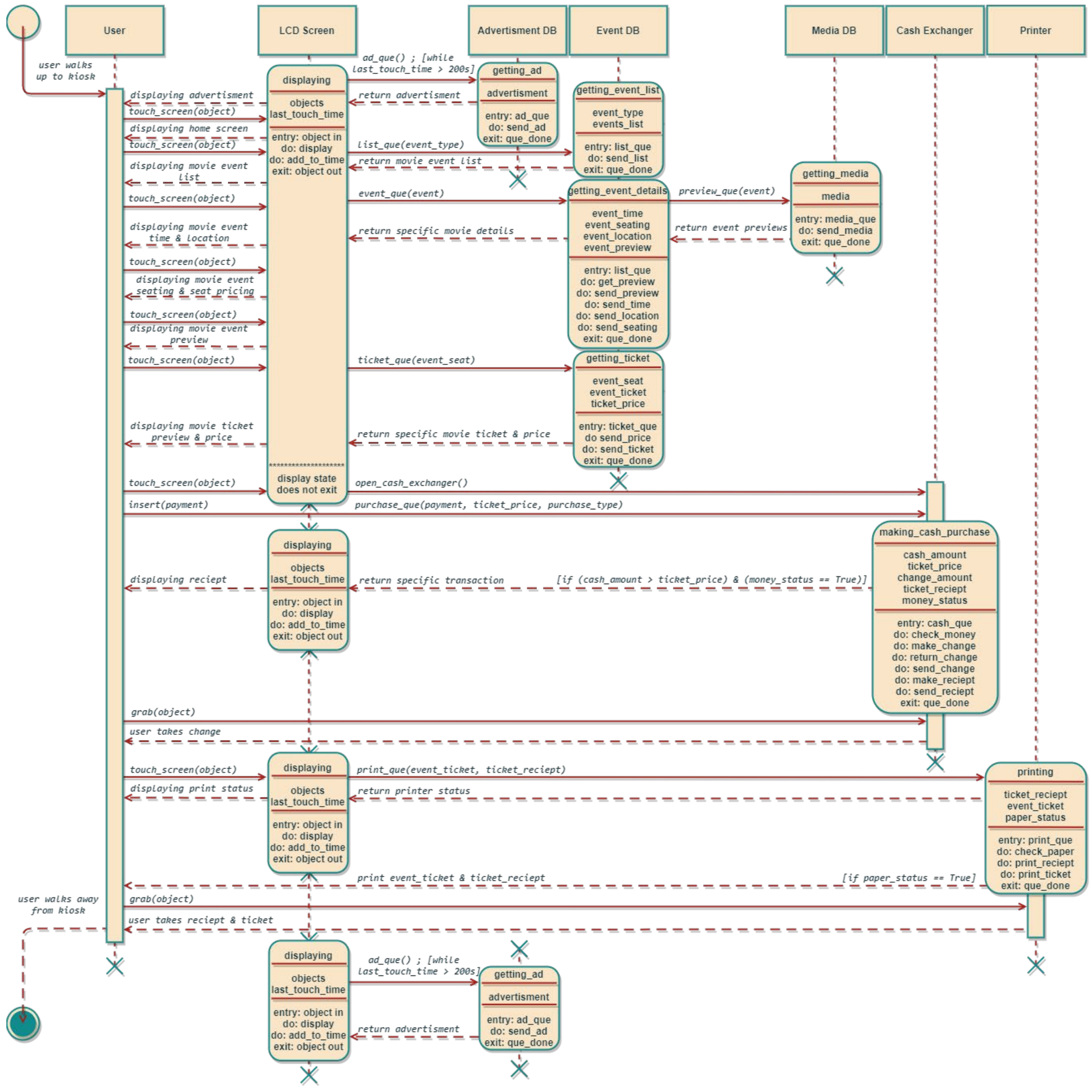
[Note]:

For all written homework in this class, a page is defined as single space, 12-point font, with one inch or less side margins, and 1.5 inch or less top and bottom margins. Write in complete sentences and provide references with citations as needed.

Sequence & State Diagrams

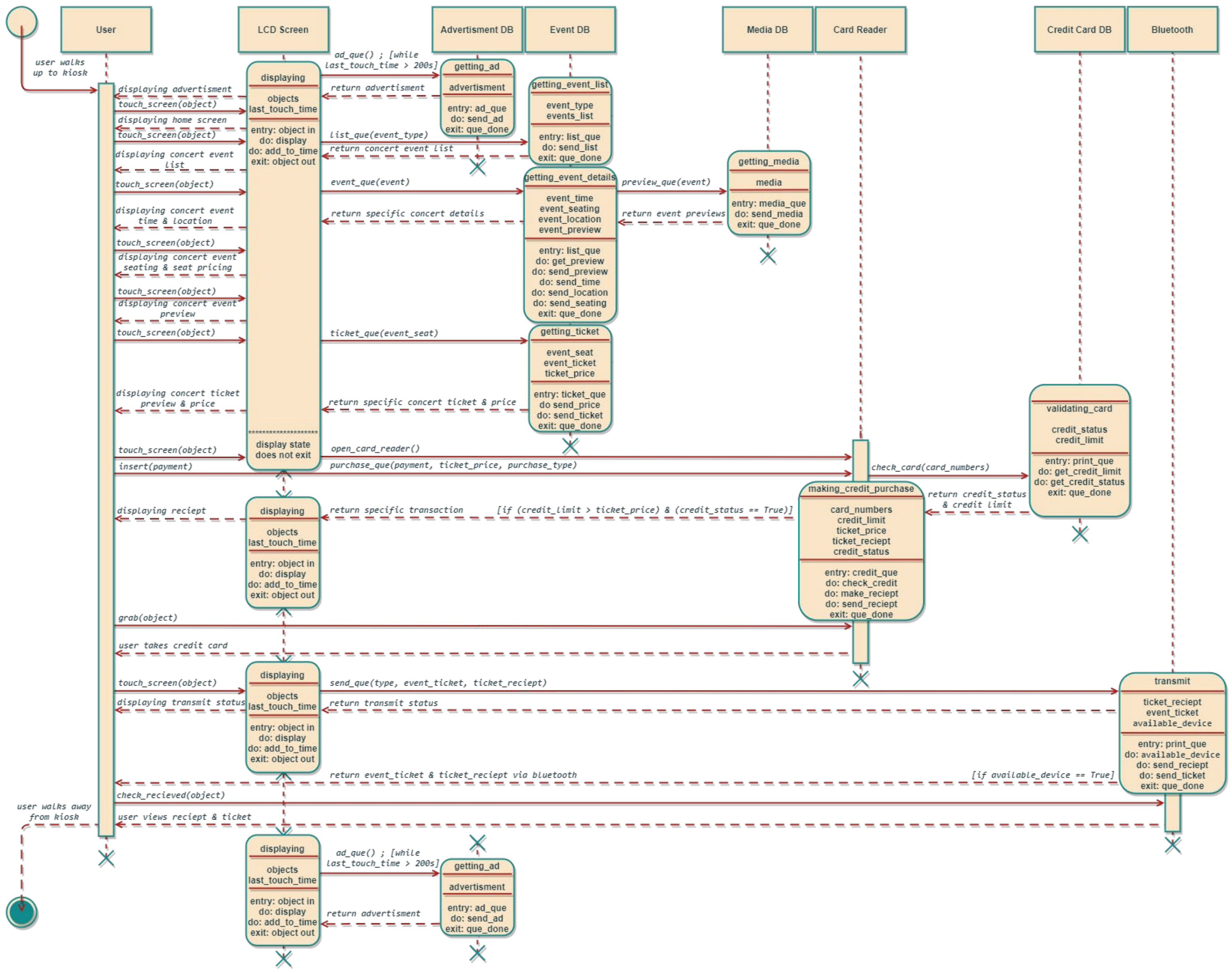
[1]

Buy Movie Ticket with Cash.



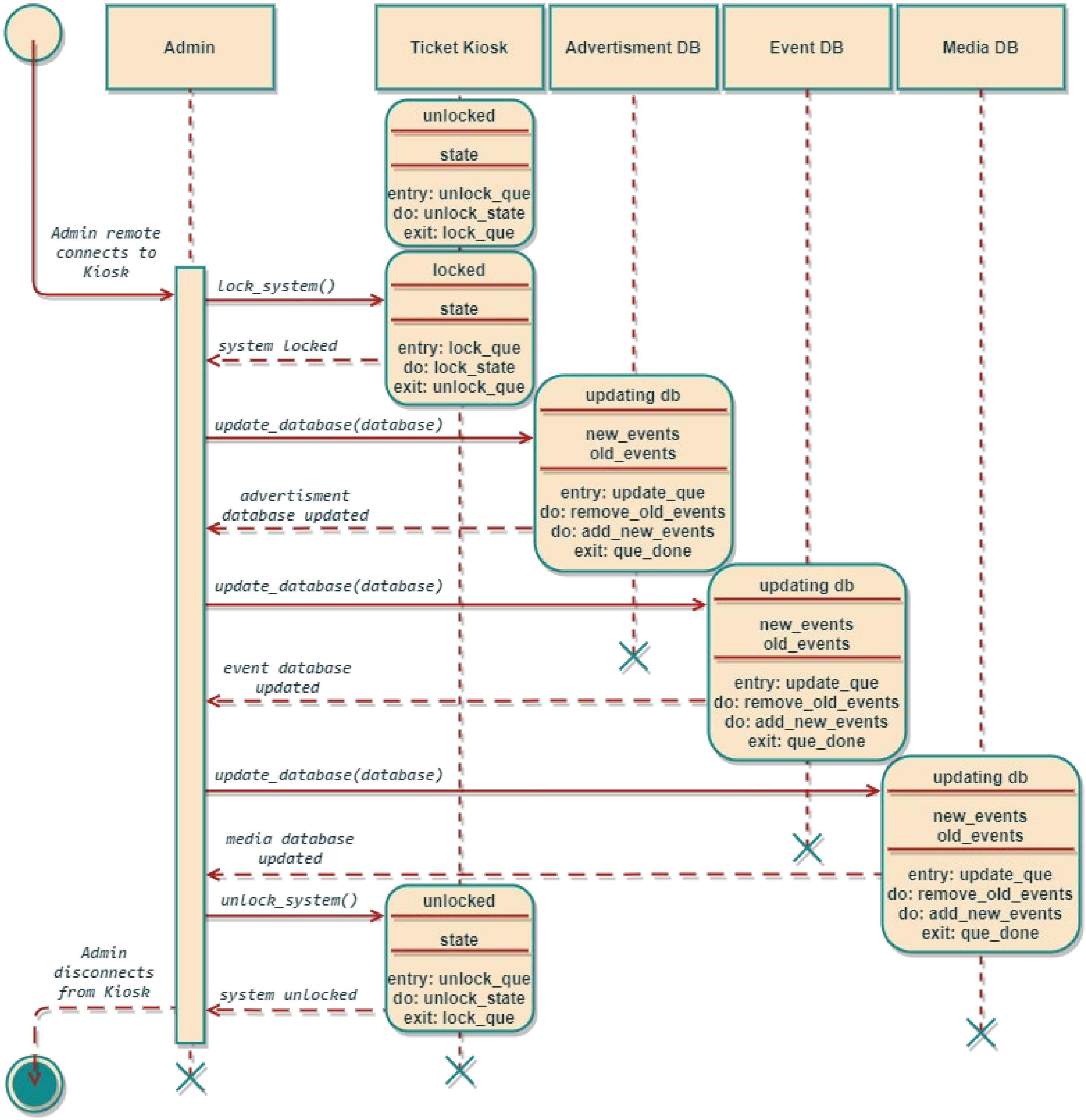
[2]

Buy Concert Ticket with Credit.



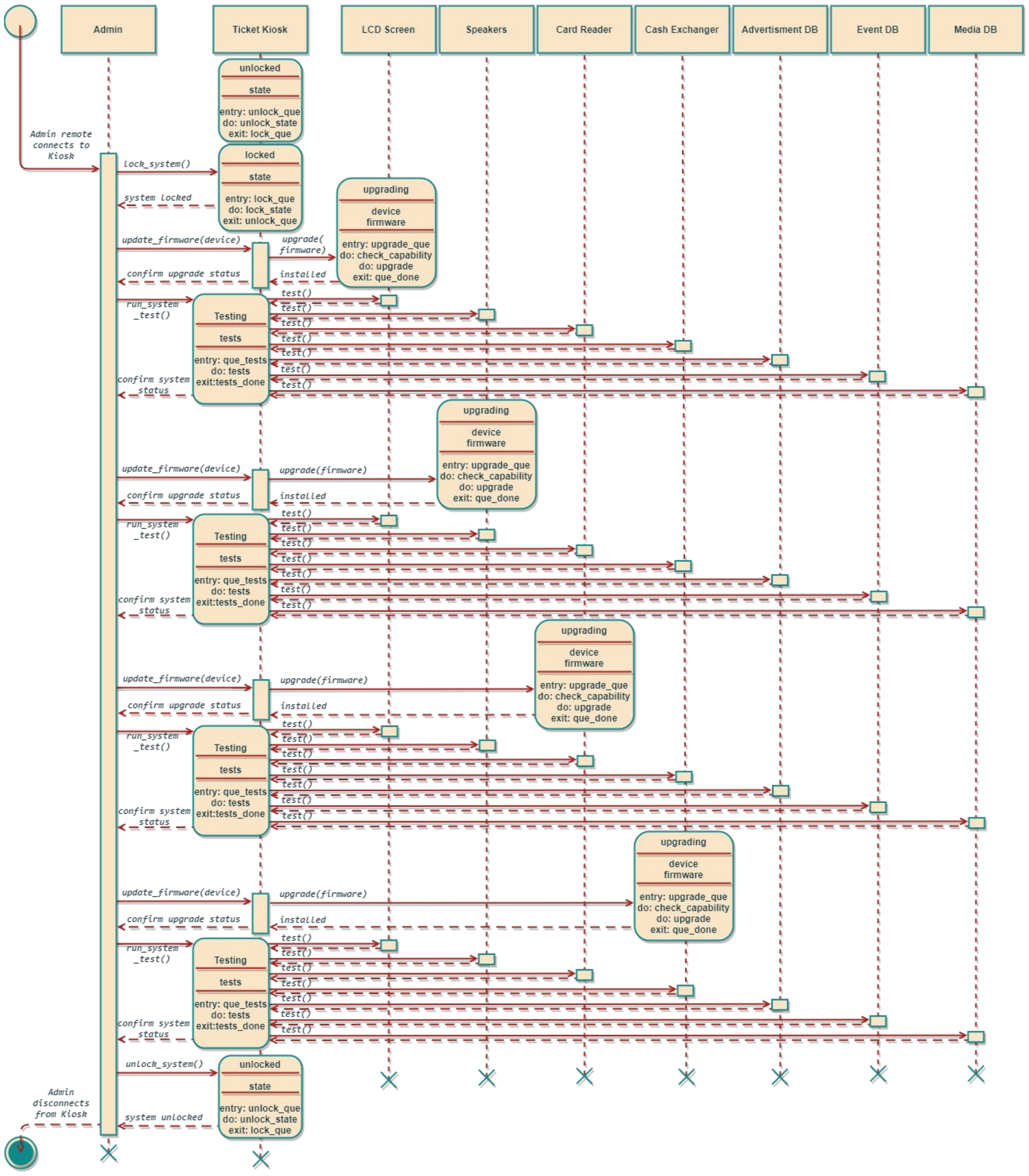
[3]

Update Events, Media, and Advertisers.



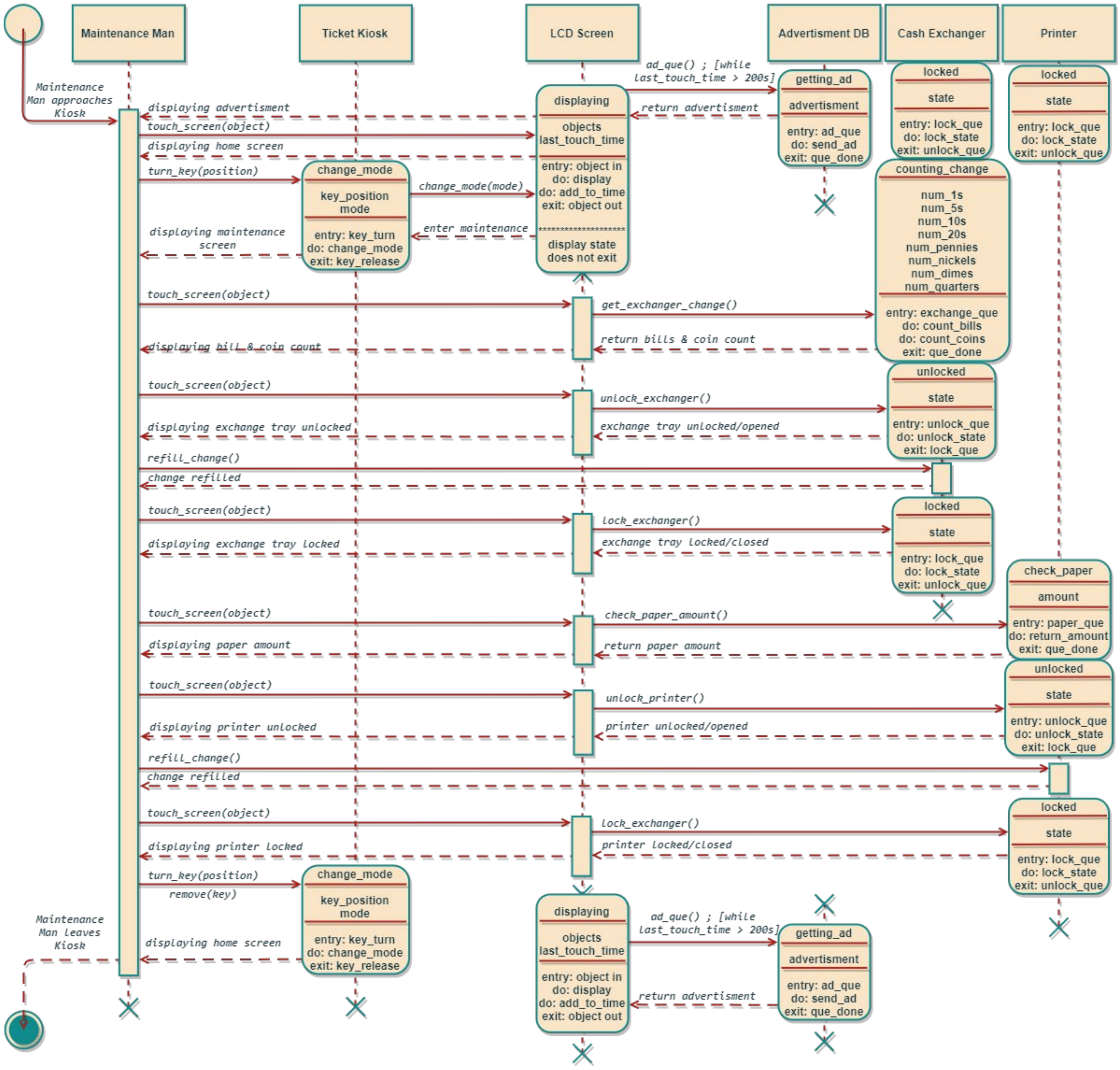
[4]

Update Events, Media, and Advertisers (Test Each Upgade).



[5]

Load Kiosk with Paper Bills and Printer Paper.



This page is intentionally left Blank.

Reuse and Component-Based Development Paper

In Michael Sparling’s reflective paper, “Lessons Learned through Six Years of Component-based Development,” he consolidates two thirds of a decade into six pages of content rich discussion. Sparling prefaces his lessons learned with a few key ideas; the computing and software field is everchanging and technologically advancing, in order to get business to adopt technological advances it must be instilled within their values, and there will always be pushback from some individuals who would rather stick with the old ways. Following this introduction, Sparling goes on to contextualize this evolution he has personally experienced through component-based development (cbd).

Traditional software development utilizes developers of each project to meld the software into the specifications, doing so often producing highly custom software while requiring large amounts of bug-chasing and redesign. Utilizing cbd makes companies value reusability and the functionality of components down the road. By taking this component approach developers are driven to produce more robust code that can be used in many applications without modification. When transitioning into cbd it is important that business describes their intentions and the foreseen benefits to keep coherence within the development cycle. Early stages will not be the same as late stages and will require lots of leg work to build a repository of components for future use. To be efficient in doing so, the business must standardize into a component framework. Sparling calls this a component reference model and highlights the need to standardize goals, design, modeling, analysis, deployment, testing, and documentation (2000, 49). Some bad habits developers exhibit within cbd is the drive to build everything as a reusable component. Rather than this reusability should only be implemented given there is a business advantage to doing so, or better said given that it will be reused. Common within developers is a pride on producing many lines of code, referenced as a metric of effort and success. This mentality can drive the coding team to create new components, effectively wasting time when there is a viable component available within the repository. By clearly identifying the reasoning a business is implementing reuse, they should be able to flush out emerging bad habits.

Having individuals that know these bad habits follow up with developers though frequent meetings in an agile environment provided additional security to productivity. Cbd also provides the ability to compartmentalize development teams based on component. Referred to as parallel development, the implementation of such a task requires a high level of communication. Those architecting the software must give clear specifications to the developers as to how their component should work. Architects should be involved with the teams to ensure that integration is possible when merging each development teams parallel progress to the final product. Implementing prototypes allows for project expectations and requirements to be set early in the development stage, while additionally involving architects for review.

Sparling concludes his article specifying some of the frameworks that have worked well within his company. With components being reused, there is a need to limit changes being made to them. The process of doing so should be clear and transparent to the teams using a component being modified. Versioning is a great way to utilize changes made to components, repositories have evolved to support such changes, what is crucial is a communication method to update those using the component when changes are made. Within the component, clear messages should be available to help those developers implement a reuse component. These are documentation standards that should be set in the component reference model, Sparling suggests in this context more is more. Similarly, there is a need to test components in use, there should be a strategy in place for doing so. As a last point Sparling correlates cbd with his company's success, and like learning any new skill, advises the importance of an iterative approach. With each attempt new information and insights will be learned and the process will be refined.

The ideas Sparling has introduced provide depth into how technology evolves within industry and enterprise. To a new software engineer, understanding the evolutionary process is crucial. It is easy for any individual to get set in certain ways. As human nature we all fall into ideas and form habits. Sparling expands to the need for open mindedness within industry and elaborates upon the benefits he has obtained from it. In contrast, the close minded often waste time and create redundancy. Sparling introduces the lines of code metric and the habit to make everything reusable to highlight. As a new software engineer, one needs to be aware of the common pitfalls that are in the field. Being able to provide business value consistently and minimize the waste from your work will look great to your managers.

Whether working for a company, developing as a hobby, or stating your own business, knowing the benefits of cbd gives you a leg up. If developing in a situation where cbd is not implemented prior, one does not need approval to start implementing. Efficient programmers know that in most all situations, the minute task they are trying to do has likely been done before. Although there may be a twist on the prior version, it is often able to be modified to fit your application. Given the code is something you will reuse, standardize the snippet, write a brief description as a reminder, and save it for future reference.

I simply cannot emphasize the amount of time saved this method has provided me personally. Many situations I deal with at work require some sort of data piping, formatting, or information retrieval. The common approach within the company has been to do by hand, taking a long time and frying your mind with the tedious task. In my case I have been able to do a few google searches, pull a couple of code snippets together, and create a short 1–3-line cmd line or powershell script to do the job for me. In general, this is not the standard process, to my understanding because nobody has done so before. Following the processes explained by Sparling, it is highly effective to prototype fast and often. In my experience I follow directly with the testing activity and iterate until I have verified functionality. In the end, 30 minutes into a 1-hour job I have completed a simple script to optimize my task, total job take 45 minutes first time around and 15 thereafter, time to complete job is quartered effectively.

In today’s remote enable environment there is enormous upside to a new software engineer to be able to do so. For example, particularly dubious individual could get away with not explaining their secret and leave the others to think they are using only methods. In my case, I have open sourced my scripts to the coworkers in my department, although I could spend 45 minutes of each time the task is completed kicking it. In any case, I am a strong advocate to start building a personal repo for reuse. Even if doing so within following business technicalities, having a repo for reference will help you create functional programs in languages you may have little to no experience in.

For those familiar with languages, they know most all languages have components / modules / packages built in. Just about every language, and even operating systems have package managers. Knowing about reuse in depth also allows to minimize unused components within a system. A simple example of this is compiling a barebones kernel of gentoo linux. As a result the user will also gain understanding of what packages are running on their system and why they are needed. High level programming languages such as python provide modules for low level system interaction such as os, subprocess, etc. Modules can also provide higher levels of abstraction such as pandas, tensorflow, etc. In summary cbd is already fermenting into the software melting pot and as an upcoming software engineer it may as well be viewed as a requirement. Taking the time to learn about the commonly used components will not only expand understanding of the programming language, but the overarching development framework, interoperability between data types, nuances, and common trolls present within the software industry.

Works Cited

Pressman, R., and B. Maxim, *Software Engineering: A Practitioner's Approach, Eighth Edition*, McGraw-Hill, 2015.

Ramachandran, M. "Software Reuse Guidelines." *ACM SIGSOFT Software Engineering Notes* 30.3 (2005): 1. Web.

Sparling, M. "Lessons Learned through Six Years of Component-based Development." *Communications of the ACM*43.10 (2000): 47-53. Web.