Software Project Management Plan

UMScheduler

Samuel Lim, Zach Zoltek, Alivia Dutcher, Yazdan Riazi

Updated March 9, 2020

1 Document Information

1.1 History

Version	Date	Description
0.1.0	8 March 2020	Initial version of plan

1.2 Document Storage

This document is developed through the online tool Overleaf. Its source is freely available at **Group 14's GitHub page** at the time of this writing.

1.3 Document Owner

The current project manager, Samuel Lim, is responsible for the development and maintenance of this document with accordance to the project group.

Contents

2 Overview

2.1 Purpose and Scope

- The current system of scheduling has little foundation for communicating or saving schedules until all information is formalized for the semester simultaneously. This implementation unfortunately leads to an unnecessary wastage of resources and effort invested on the part of advisors.
- For certain faculty, this results in a feedback process too far into the semester to make significant adjustments. The needs of persons involved should be adequately accommodated directly at the time of conceptualization.
- The purpose of this project is to provide an alternative outlet to coordinating and generating course schedules for advising that allows for transparent and management of schedule conflicts and change of priorities in a timely manner.

2.1.1 Notable Scope

- A user's guide and a system installation guide will be provided. One-on-one personalized training isn't one of the project deliverables.

2.2 Goals and Objectives

Goals and objectives define expected project outcomes. Goals are broad and inspirational. Objectives are narrow and measurable. Project goals generally related project outcomes to business objectives (reduced cost, increased revenue, improved quality, etc). A well-worded objective is SMART: Specific, Measurable, Attainable/Achievable, Realistic and Time-bound.

1. The primary result of the project is to present a usable log to advisors, namely Gina Campbell, in efforts to interact with informal scheduling information in a structured manner.
2. The project should be easy to access and provide additional information should a session increase in scale without being cumbersome to the user to navigate, as accessible through a web portal.
3. Using the tentative technologies (listed below), documentation will be a combination of automation and manual explanation, allowing for a natural dynamic from user-facing to computationally engaged components.
4. Additionally, conflicts in generated schedules should be clear and straightforward to diagnose and avoid if possible. The primary result should emphasize these features in any relevant presented interaction. Based on our initial architecture, editing professors and their constraints should not be an issue concerning rewrites of the program. The session configuration can be edited by users, and eventually administrative roles distinctly (reach goal).
5. As mentioned in the section above, the delivered product should demonstrate the ability* to deliver content tailored to the user's preferences,

*but a complete implementation of this feature is beyond the scope of the current project.

- Due to the limits of the course constraining the project, many non-essential features will need to be condensed to their minimal presentable form. High-priority features dealing with structural logic and defining ideal user-facing interaction will be maintained early on the process of iteration.

2.3 Deliverables

The following items will be delivered to Professor Bingham and Gina Campbell on or before the conclusion of the course:

1. Source code of the scheduling service and its containing web application
2. Requirements document
3. User's guide
4. Test Plan
5. System test cases
6. Suite of regression tests

2.4 Assumptions and Constraints

Assumptions:

1. The serving API works on the test hardware.
2. A senior engineer will be assigned to the project during the first 4 weeks.
3. The open server and deployment dispatcher can be registered and delivered by March 31.
4. Any database within the scope of a student or professional is allowed for the project.

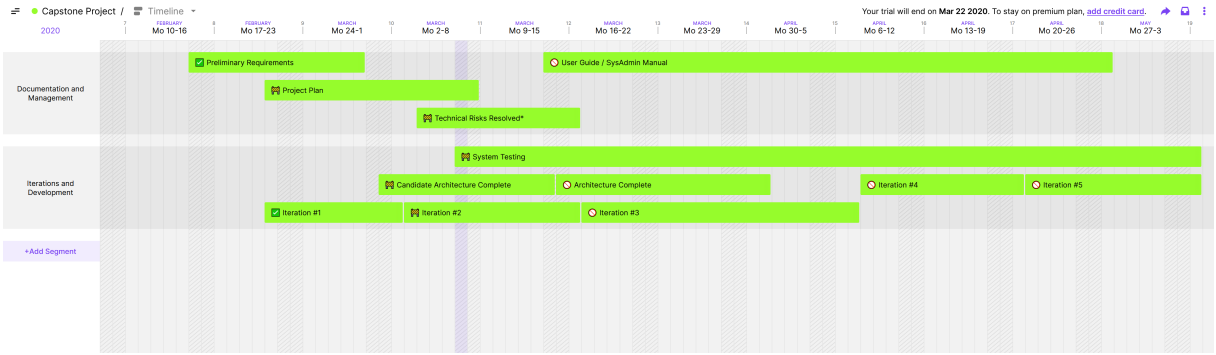
Constraints:

1. The software must run on at least a mobile device housing Chrome, Firefox, Safari, or Brave.
2. The database must be open source.
3. The software must be ready by May 15.
4. The scheduling format should be downloadable as a spreadsheet and shareable.

2.5 Schedule and Budget Summary

The schedule summary shows start and end dates for high-level activities ending in major milestones or deliverables. Milestones are major events in the project life cycle that are used to measure progress.

A Gantt chart is an excellent tool for visualizing the start and stop dates of major scheduled activities.



The budget summary shows total project cost, possibly broken down into separate categories for such things as salaries, equipment, travel, overhead, etc.

2.5.1 Personnel

Role	Hired	hours/week	# weeks	Hours	Rate (per hour)	**Total**
Project Manager	1	6	14	84	$100.00	$8400
Requirements Engineer	1	6	14	84	$85.00	$7140
Software Engineer	2	6	14	168	$65.00	$10920
Total	4	24	14	336	(avg.) $78.75	$26460

2.5.2 Other Costs

These include equipment, travel, overhead, and deployment costs.

Due to the nature of the course, travel costs will be treated as no cost for the scope of the semester. Equipment and overhead costs are specified only under the build times necessary for GitHub integration and deployment are available at no cost to open-source projects and in some cases, to anything hosted on the site.

2.6 Success Criteria

- Total project cost does not exceed 20% of the post-requirements phase estimate.
- All high-priority use cases in the requirements specification are delivered before May 15.
- The project must have a minimum working example (MWE) before March 31.
- The project is buildable at any major release beyond the MWE until May 15.

2.7 Definitions

This section should define potentially unfamiliar or ambiguous words, acronyms and abbreviations.

1. **UMKC Scheduler/UMScheduler**: the product that is being described here; the software system specified in this document.
2. **System**: the current product under assessment described as above.
3. **Use case**: describes a goal-oriented interaction between the system and an actor. A use case may define several variants called scenarios that result in different paths through the use case and usually different outcomes.
4. **User**: the person or persons who will actually interact with the UMKC Scheduler. Scenario: one path through a use case
5. **Actor**: user or other software system that receives value from a use case.
6. **Project**: activities that will lead to the production of the product described here. Project issues are described in a separate project plan.
7. **Developer**: the person or organization developing the system, also sometimes called the supplier.
8. **Shall**: adverb used to indicate importance; indicates the requirement is mandatory. "Must" and "will" are synonyms for "shall".
9. **Should**: adverb used to indicate importance; indicates the requirement is desired but not mandatory.
10. **May**: adverb used to indicate an option. For example, "The system may be taken offline for up to one hour every evening for maintenance." Not used to express a requirement, but rather to specifically allow an option.
11. **Controls**: the individual elements of a user interface such as buttons and check-boxes.
12. **App**: a piece of software designating the scope of the project. "Application" is a synonym for "app".

2.8 Plan Evolution

Before the start of an iteration, the project plan will be updated to include a schedule of detailed tasks for the upcoming iteration. At the conclusion of an iteration, the project plan will be updated to include the actual effort for each completed task.

Risk mitigation efforts will be evaluated at the start of each iteration. Severe risks will be analyzed and added to the project plan as soon as they materialize.

3 Startup Plan

3.1 Team Organization

3.1.1 Project Manager

- The project manager is responsible for creating the project plan (with input from those doing the work), managing risks, running the weekly

team meeting and providing monthly status reports to senior management.

3.1.2 Software Engineers (2)

- The software engineers are primary responsible for coding and unit testing modules. They are also expected to take part in architecture planning and review meetings.

3.1.3 Requirements Engineer

- The requirements engineer is responsible for the requirements, assessment and advancement of specifications and technical requirements, coordinating with the team to document further requirements.

3.2 Project Communication

As stated previously in the document, the project administration will be handled both online and in-person.

3.3 Technical Process

3.4 Tools

- Programming Language Stack - Rust, WebAssembly available as a compilation target (separate language), and JavaScript (with TypeScript) to bootstrap the system.
- Version Control - Github (Git) and local Git hosting
- Bug tracking and triage will be registered through GitHub issues
- Build tools will be available under the Rust ecosystem with `cargo`
- Testing will also be through `cargo`'s testing suite and macro language.
- Integration and Deployment will be held locally on a server and also set up on GitHub with Travis CI and GitHub Actions.

4 Work Plan

4.1 Activities and Tasks

4.1.1 Feature: Scheduling Constraints

In order to generate a schedule effectively, the system requires constraints in the form of user inputted rules. This use case defines the process by which a faculty member might input a rule for the system to take into account when generating a schedule.

- Cost: High
- Risk: High
- Value: High

Use Case: Add Rules for Constraints

Basic Path:
1. User navigates to UMKC scheduler website
2. User logs in to their account
3. System prompts user to: Create New Schedule or View Previous Schedules
4. User chooses Create New Schedule
5. System gives user all rule generating options
6. User enters their constraints for current semester

Additional Requirements

Users should be able to add and edit additional rules for each schedule as well as fork off previously generated schedules in some form to create new schedules from existing information.

4.1.2 Feature: Profiles

Description and Priority

Users should have separate profiles that they login into. For each profile, there should be persistent memory of previously generated schedules. User should be able to view previously rendered schedules.

- Cost: medium
- Risk: high
- Value: high

Use Case: View and Share Previous Schedules

Basic Path:
1. User navigates to UMKC scheduler website
2. User authenticates to access profile
3. System prompts user to: Create New Schedule or View Previous Schedules
4. User chooses View Previous Schedule
5. System displays a database of previously rendered and store schedules

Additional Requirements

Users should be able to share schedules publicly without breaching security protocols (sharing via a link without editing capabilities).

4.1.3 Feature: Schedules

Description and Priority

After the system receives rules from the user, the system will generate a sample schedule. Schedule generation should eliminate minimal conflicts and display remaining conflicts when rendered.

- Cost: high

- Risk: high
- Value: high

Use Case: Schedule Generation

Basic Path:

1. User navigates to UMKC scheduler website
2. User logs in to their account
3. System prompts user to: Create New Schedule or View Previous Schedules
4. User chooses Create New Schedule
5. System gives user all rule generating options
6. User enters their constraints for current semester
7. User selects Generate Schedule option
8. System renders a schedule and displays it to the user

4.2 Release Plan

4.2.1 Releases and Work Products

Date (MM/DD 2020)	Task Description
02/21	Project Charter Approved
02/28	Product Feature Set Baselined
02/29	Preliminary Requirements Complete
03/02	Iteration #1 Complete
03/06	Preliminary Project Plan Complete
03/14	Candidate Architecture Complete
03/16	Technical Risks Resolved*
03/16	Iteration #2 Complete
03/31	Architecture Complete
04/07	Iteration #3 Complete
04/20	Iteration #4 Complete
04/27	User Guide & System Administration Manual
05/04	Iteration #5 Complete
05/04	System Test Complete
05/04	Product Released

4.3 Iteration Plans

Iterations shall be coordinated as a matter of recourse between previous development and requirements for the current sprint. Weekly meetings will designate and update optimal paths for development or requirement re-assessment. The end of each iteration will end with a recap meeting, reflecting on the ups and downs of the previous sprint and how best to move forward.

4.4 Budget

The project shall stay with the means of the actors involved. Ideally, the resulting product should not scale further in cost than the current implementation within the administrative offices. Including maintenance of service, the uptime and access of the product should stay within the bounds of $800 initially, with respect to the advantages of open-source facilities.

5 Control Plan

5.1 Monitoring and Control

For formal dates of review deadlines, see the figure below. In addition to the major task revisions and monitoring established, weekly meetings will take place at least twice a week expanding on development progress and documentation updates, on Monday, Wednesday, or Friday.

Date	Description
–	Project Charter Reviewed
–	Product Feature Set Baselined
–	Preliminary Requirements Review
03/10	Candidate Architecture Reviewed
03/12	Technical Risks Reviewed*
03/26	Formal Architecture Reviewed
04/20	User Guide & System Administration Manual
05/01	System Test Reviewed

5.2 Metrics Collection

Phase	Measurement	Source
Release Planning	Record effort estimates for product features	P.M.
Iteration Planning	Record effort estimates for scheduled tasks Update effort estimates for product features Update estimated dates in release plan	P.M.
Iteration Closeout	Record actual effort for scheduled tasks Record actual effort for product features Record LOC count for modules written	P.M./S.W.E.
System Test	Record the rate at which errors are found.	S.Q.A.
Project Closeout	Archive project performance data in process database. (See process database definition for a list of measures to record.)	P.M.
Ongoing	Record defects found from integration testing through the first of release. Assign each defect to one of the following: blocked, critical, major, minor, trivial. Keep track of the current status: open, assigned, fixed, closed.	P.M./S.W.E./S.Q.A.

6 Supporting Process Plans

6.1 Risk Management

As previously referenced in the project charter:

- A major risk that the team faces is that a majority of the members lack of experience in the Rust language. Due diligence should be maintained when developing under this environment and when assessing further outcomes in future iterations.
- This generates considerable unknowns when (1) making estimates, (2) choosing related technologies, (3) deciding on implementations, and (4) adhering to best practices.
- Another obstacle is the limited time each member can contribute each week, meaning work done will have less effective time for implementation as the project grows in complexity.

6.2 Configuration Management

1. All work products will be stored in a centralized CVS repository running on a central server.
2. The naming convention for documents will be: `MMM.GGG.NNN.suffix` where `NNN` is a mnemonic that reflects the function of the document, `GGG` is a label representing the unique group leading the project, `MMM` is the course or supervising group managing the objective assessment of the project's progress, and `suffix` is the standard/normal suffix for the document type. For example, the second version of the requirements document created might be labeled: `CS451R.Group14.RequirementsDocument.docx`.
3. All project (work products) items (documents, source code, test cases, program data, test data, etc) will be stored in the CVS repository but not all will be under change control (subject to formal change control procedures.) Only the system requirements, project plan and source code will be baselined and under configuration control.
4. Items that are subject to change control will be considered baselined after a group review at the end of the life cycle phase during which they are created. Baselined here means that the product has undergone a formal review and can only be changed through the prescribed change control procedures.
5. The change control procedure once a product is baselined is: (1) anyone wanting to make a change to a baselined item sends an email to the rest of the group describing the change, reason for the change, expected impact, and timeline for integrating the change. (2) if no one responds to the group within 2 days with a reason for why the change request shouldn't be permitted, it will be considered accepted and the person proposing the change may proceed with the change. If anyone does object to the change,

the reason for objecting will be discussed at a meeting where everyone is invited to attend and voice their opinion. At the end of the meeting a democratic vote will be held to decide whether or not the change should be allowed.

6. Including a change history with all documents is encouraged but only required for baselined documents. The change history should be at the front of the work item and include: (1) the name of the person making the change, (2) brief description of what has changed, (3) reason for the change, and (4) the date the change was integrated.

6.3 Verification and Validation

Due to the non-traditional technology stack of this project, verification and validation will be assessed at the level of formal verification wherever possible or already defined, and traditional test acceptance metrics will be used otherwise, which may be defined as seen in boundary analysis in theorem provers or static analysis of the resulting schedule constraints. Further specifications may be specified in another document directed toward verification and validation requirements in the future.

6.4 Product Acceptance

The product must be available as a standalone web application without download requirements on all major modern browsers supporting WebAssembly. The product should also be available during all office hours whenever possible.