A Recipe for Web Apps

Lewis Lloyd @LloydTao
Lead Developer
Magpie Education

Introduction

Hatless Studios (2019 - 2020)

- Software Engineer
- Implementing, testing and documenting software
- Key role in branching strategies and code quality assurance

Magpie Education (2020 - Present)

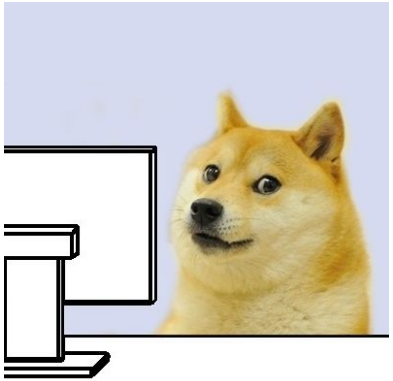
- Lead Developer
- Technical strategy and product management
- Full-stack web development (AngularJS, Django, AWS)

And also, a third-year MSci Computer Science student at the University of Exeter!

Agenda

What are we learning about?

- Running a successful software project!



Project Ingredients

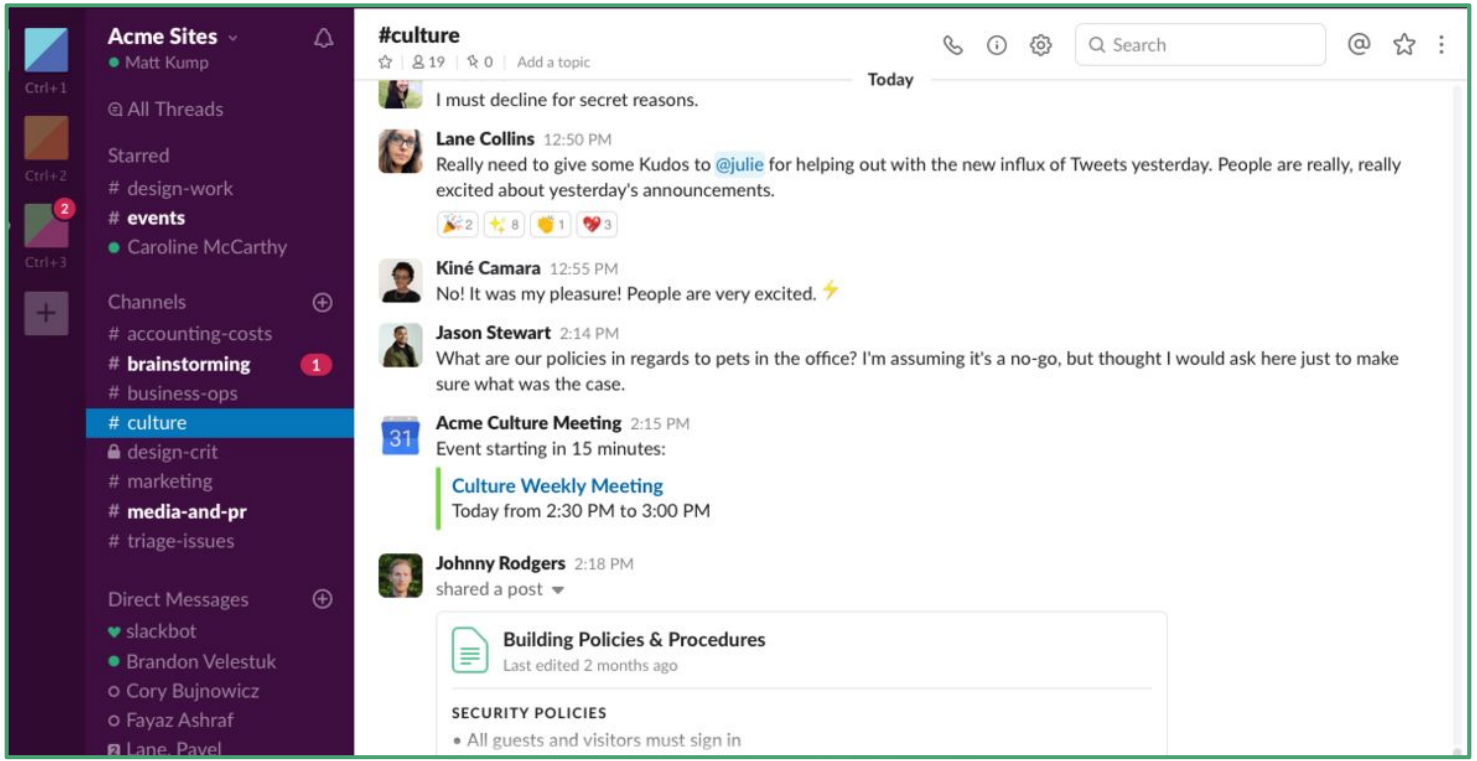
1. Communications
   How do I keep in touch with my teammates?
2. Agile Board
   What am I working on? How do I make my progress visible?
3. Code Repository
   Where do I store my code? How do we collaborate?
4. Deployment
   How do I get my web app online? How do I keep it updated?

1. Communications: Slack

Slack is an incredibly popular communications platform, with 10+ million daily active users.

It’s super easy to search through chats and shared files, compared to apps like *Facebook Messenger*.

1. Communications: Slack



1. Communications: Slack

How to use it effectively:

- Organise conversations into **project**-level, **team**-level and **feature**-level chatrooms.
- Keep relevant conversations within threads to avoid *message clutter*.

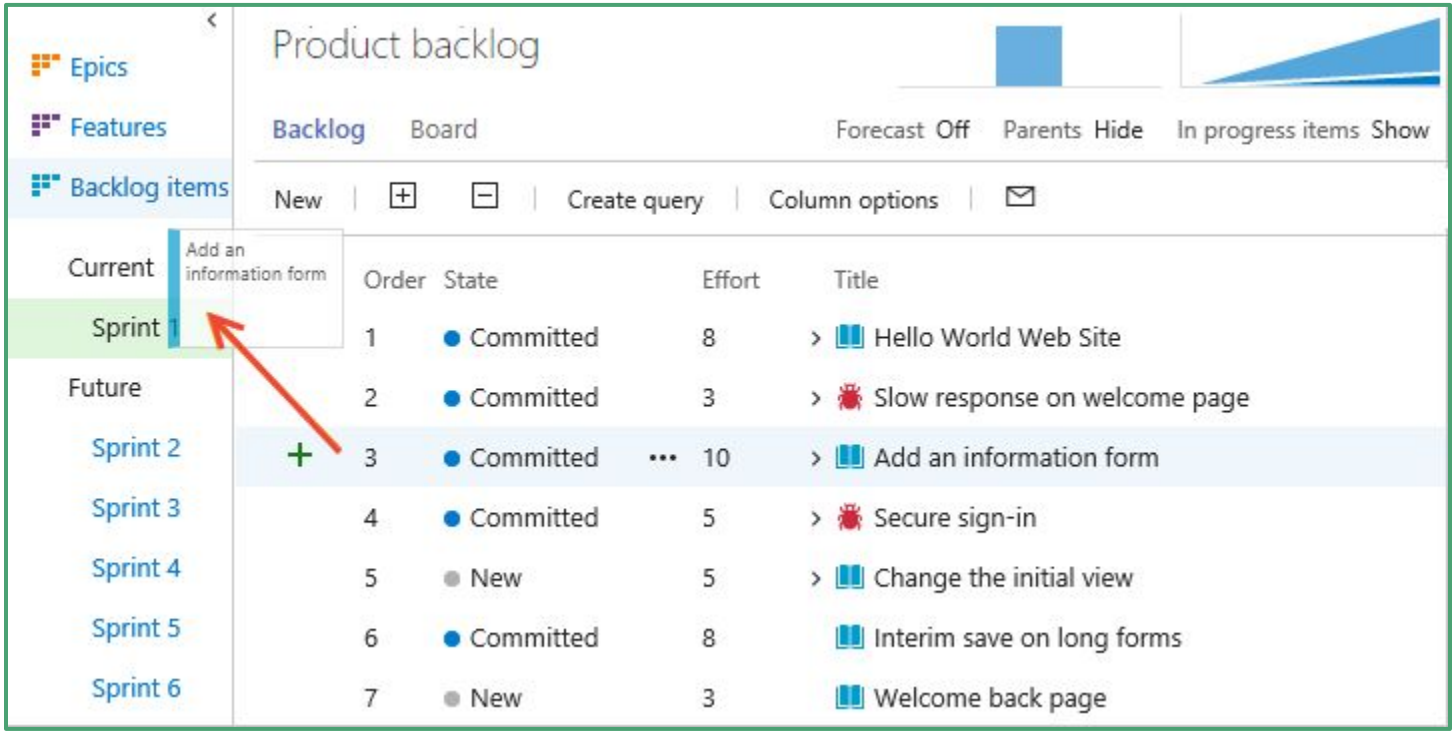
2. Agile Board: Azure DevOps

Azure DevOps is a piece of *Microsoft* software.

Version control, automated builds, testing and release management capabilities are in the suite.

We're just interested in the **agile board**.

2. Agile Board: Azure DevOps



2. Agile Board: Azure DevOps

A large part of your **initial** project planning will be breaking down your project into **epics**, **features** and **user stories**.

Be smart about what go on to the board!

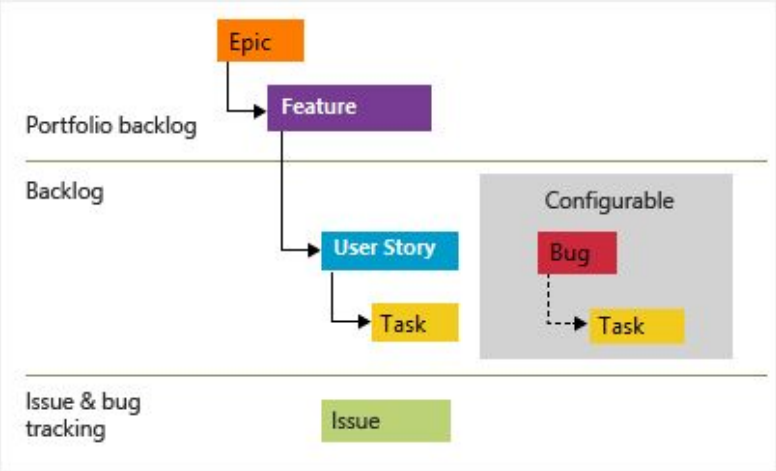
Scope everything out, and clearly define every task.

- Do not start your project's implementation until your team is happy with its scope and definitions.
- Do not start a task unless it is clearly defined.
- If not, pull the team together to discuss re-defining a task.

2. Agile Board: Azure DevOps

Microsoft has a great piece on defining features and epics, and organising backlog items.

https://docs.microsoft.com/en-us/azure/devops/boards/backlogs/define-features-epics



3. Repository: GitHub

Repository setup:

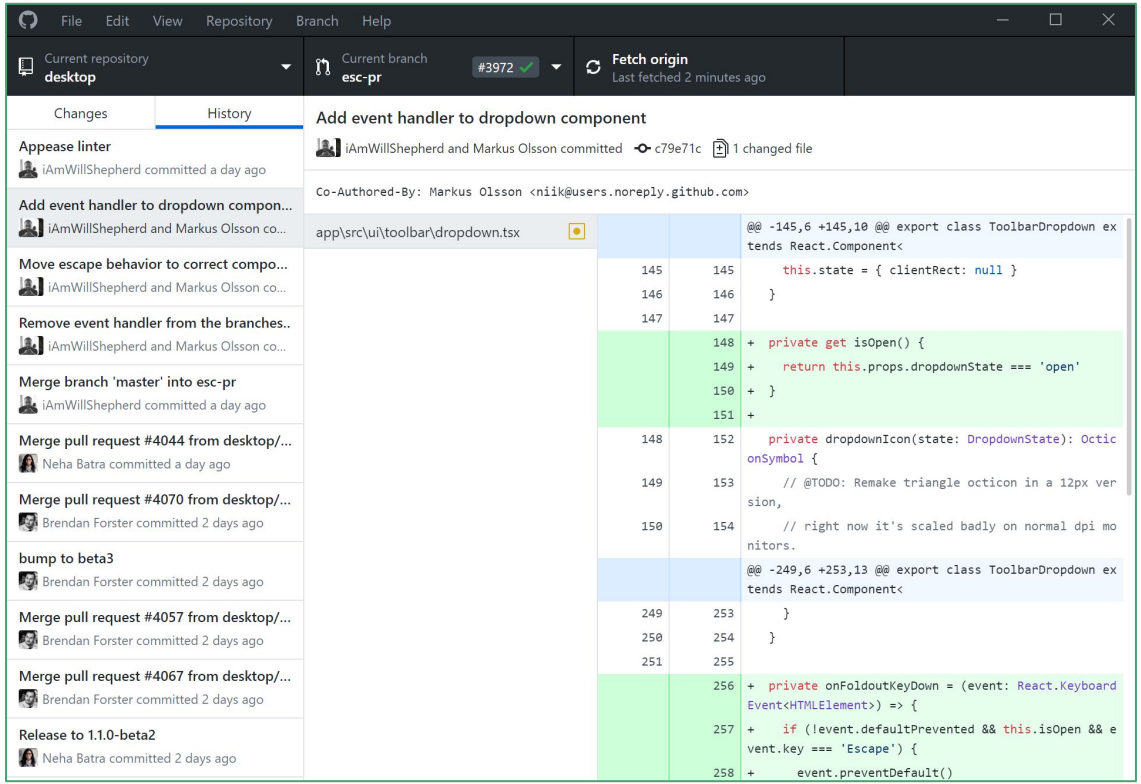
1. Create a *GitHub* repository.
2. Make sure that everyone is added as a collaborator.
3. Make sure that everyone has cloned it, and knows how to fetch, pull and push commits.

3. Repository: GitHub Desktop

You need a *Git* client in order to access a *Git* repository.

Using a GUI, rather than command line, is much easier.

3. Repository: GitHub Desktop

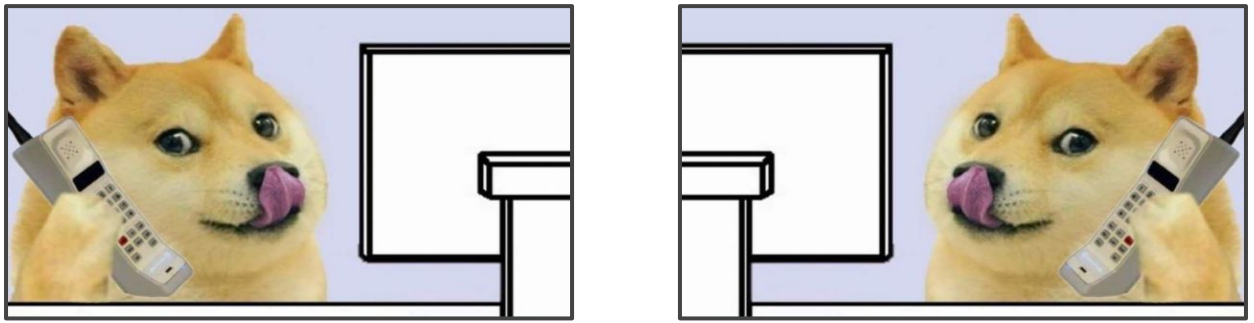


Bonus: Collaboration

We’ve solved file sharing, but we have a new problem.

Code is fragile.

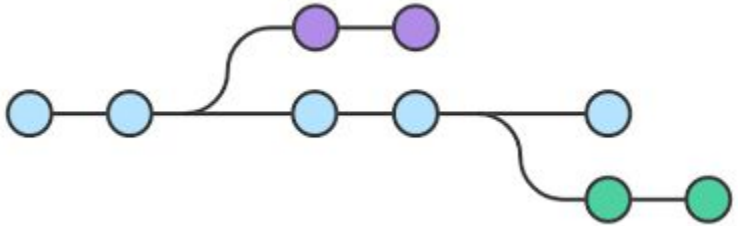
How can remote collaborators ensure code integrity?



Collaboration: Branches

We need to avoid conflicts in two places: **during** and **after** development.

- To avoid conflicts **while** developing, we use branches.
- This allows us to push our changes to the centralised repository without overwriting or breaking someone else's changes.



Collaboration: Branching Strategy

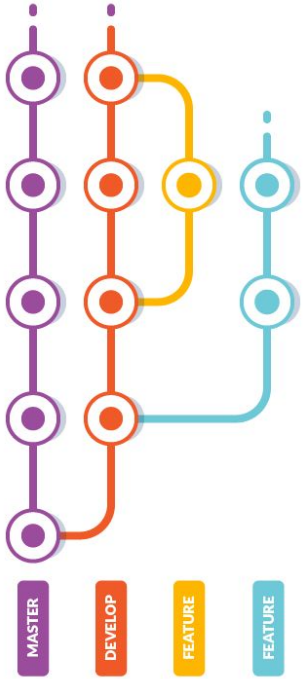
We need to branch in a smart way.

1. Branch off of the default branch for any new feature.
2. Build the feature, without going out of scope.
3. Launch a **pull request**, to get your branch (and its commits) reviewed and merged back into default.

If a branch goes out of date, merge the default branch into it.

Once merged, the branch should be deleted.

Don’t let branches get too old!

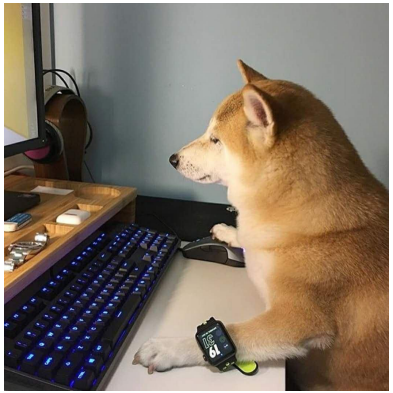


4. Deployment

Deployment might seem like a final step, but you should actually do it **straight away!**

- Get a version of your web app up that's just your initial project files.
- Allows you to figure out your process for actually getting an app deployed.
- New features and bug fixes are constantly rolled out as pull requests are approved and merged.

CI/CD is not something you want to figure out in crunch time!



4. Deployment - Elastic Beanstalk (AWS)

Elastic Beanstalk is a complete solution for deploying database-driven web apps.

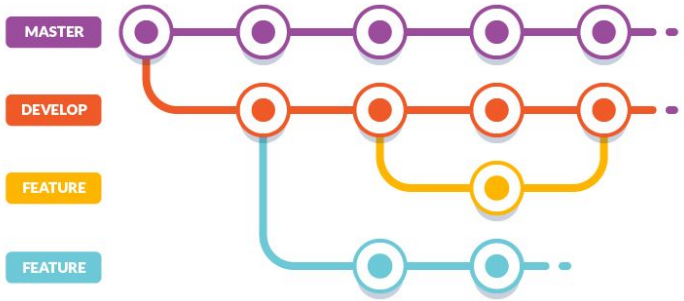
- This includes the SSL, load balancing, compute cloud provisioning and database.
- Bad application versions will automatically roll back.
- Any standard usage (EC2, S3, Load Balancing) should all fall under free tier.

If you want to go the extra mile, *AWS CodePipeline* can integrate *GitHub* with *Elastic Beanstalk*.

4. Deployment: Branching Strategy

Your branching strategy is integral for stable releases.

- The *default* branch remains in a production-ready state.
- Each merge introduces feature-complete code.
- Any pushes to the default branch trigger a deployment pipeline.



Conclusion

1. Communications

   Keep conversations organised by using chat rooms and threads.

2. Agile Board

   Clearly define your scope and tasks. Keep progress visible by tracking it on the board.

3. Code Repository

   Set up your Git repository and collaborators. Make sure everyone knows how to Git.

4. Deployment

   Deploy your initial project early on, and keep it updated.

Questions

Any questions?

Wrapping up

Stay in touch!

- LinkedIn: Lewis Lloyd
- GitHub: LloydTao
- Twitter: LloydTao
- Instagram: LloydTao

Email: Lewis_Lloyd@live.co.uk