Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Bonn-Aachen
International Center for
Information Technology

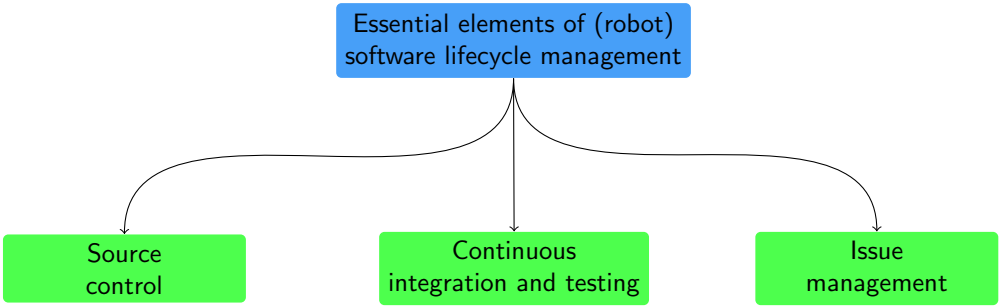
Institute for AI and
Autonomous Systems

Robot Software Development Lifecycle

Dr. Alex Mitrevski
Master of Autonomous Systems

Structure

- Source control
- Continuous integration
- Issue management


Essential elements of (robot) software lifecycle management
Source control
Continuous integration and testing
Issue management

Source Control Basics

- Source control (aka version control) is **a process of managing changes to source code files**

Source Control Basics

- Source control (aka version control) is **a process of managing changes to source code files**
- In this context, we say that **a project is stored in a repository**, which is **a collection of files under source control**

Source Control Basics

- Source control (aka version control) is **a process of managing changes to source code files**
- In this context, we say that **a project is stored in a repository**, which is **a collection of files under source control**
- There are two main repository versions that we refer to when discussing source control:

Source Control Basics

- Source control (aka version control) is **a process of managing changes to source code files**
- In this context, we say that **a project is stored in a repository**, which is **a collection of files under source control**
- There are two main repository versions that we refer to when discussing source control:
  - **Upstream version**: The version of a repository that is **hosted centrally** (e.g. on an external server) and that **everyone can access**

Source Control Basics

- Source control (aka version control) is **a process of managing changes to source code files**
- In this context, we say that **a project is stored in a repository**, which is **a collection of files under source control**
- There are two main repository versions that we refer to when discussing source control:
  - **Upstream version**: The version of a repository that is **hosted centrally** (e.g. on an external server) and that **everyone can access**
  - **Local clone**: A local copy of a repository, whose changes **may be made available** in the upstream version at a later point

Why Source Control?

Source control is **an essential element of modern software development** — in robotics and otherwise — for a multitude of reasons:

Why Source Control?

Source control is **an essential element of modern software development** — in robotics and otherwise — for a multitude of reasons:

Repository backup

Backup is one of the main motivations for using source control, both to **prevent code losses** and because **it makes it relatively easy to revert to old versions**

Why Source Control?

Source control is **an essential element of modern software development** — in robotics and otherwise — for a multitude of reasons:

Repository backup

Backup is one of the main motivations for using source control, both to **prevent code losses** and because **it makes it relatively easy to revert to old versions**

Collaborative development

Source control provides **shared access to a code repository** and an organised way of **managing the contributions of individual developers to the repository**

Why Source Control?

Source control is **an essential element of modern software development** — in robotics and otherwise — for a multitude of reasons:

Repository backup

Backup is one of the main motivations for using source control, both to **prevent code losses** and because **it makes it relatively easy to revert to old versions**

Collaborative development

Source control provides **shared access to a code repository** and an organised way of **managing the contributions of individual developers to the repository**

Preserving the development history

Source control is also a way to preserve the development history of a project — **the usefulness thereof depends on the quality of the commit messages!**

Why Source Control?

Source control is **an essential element of modern software development** — in robotics and otherwise — for a multitude of reasons:

Repository backup

Backup is one of the main motivations for using source control, both to **prevent code losses** and because **it makes it relatively easy to revert to old versions**

Collaborative development

Source control provides **shared access to a code repository** and an organised way of **managing the contributions of individual developers to the repository**

Preserving the development history

Source control is also a way to preserve the development history of a project — **the usefulness thereof depends on the quality of the commit messages!**

Code sharing and reusability

The way in which **free and open source software is accessed and can be contributed to** is also simplified through source control

Repository Interaction

- Interaction with a repository occurs through **operations** that **register file changes** or **interact with a remote repository**

Repository Interaction

- Interaction with a repository occurs through **operations** that **register file changes** or **interact with a remote repository**
- The primary operations in a repository are **staging**, **committing**, **pushing**, and **pulling**

Repository Interaction

- Interaction with a repository occurs through **operations** that **register file changes** or **interact with a remote repository**
- The primary operations in a repository are **staging**, **committing**, **pushing**, and **pulling**

Pulling

The process of **retrieving changes from the central repository** and recording those in the local repository's tree

Repository Interaction

- Interaction with a repository occurs through **operations** that **register file changes** or **interact with a remote repository**
- The primary operations in a repository are **staging**, **committing**, **pushing**, and **pulling**

Pulling

The process of **retrieving changes from the central repository** and recording those in the local repository's tree

Staging

Registers an intention to record changes to existing repository files or to **add new files to a repository**

Repository Interaction

- Interaction with a repository occurs through **operations** that **register file changes** or **interact with a remote repository**
- The primary operations in a repository are **staging**, **committing**, **pushing**, and **pulling**

Pulling

The process of **retrieving changes from the central repository** and recording those in the local repository's tree

Staging

Registers an intention to record changes to existing repository files or to **add new files to a repository**

Committing

Follows the staging process and **records file changes** to a **repository's local history tree**

Repository Interaction

- Interaction with a repository occurs through **operations** that **register file changes** or **interact with a remote repository**
- The primary operations in a repository are **staging**, **committing**, **pushing**, and **pulling**

Pulling

The process of **retrieving changes from the central repository** and recording those in the local repository's tree

Staging

Registers an intention to record changes to existing repository files or to **add new files to a repository**

Committing

Follows the staging process and **records file changes** to a **repository's local history tree**

Pushing

Records any local changes in the repository's history tree **to the central repository**

Git

- There are various source control systems, but **Git is the most widespread system**, particularly due to platforms such as GitHub and GitLab, which significantly simplify the interaction with and management of Git repositories

Git

- There are various source control systems, but **Git is the most widespread system**, particularly due to platforms such as GitHub and GitLab, which significantly simplify the interaction with and management of Git repositories
- Git is a **decentralised source control system** — every developer has **their own copy of a repository** and changes are integrated into a main repository through **merge operations**
  - This is opposed to centalised source control systems, where commits are directly added to a central repository

Git

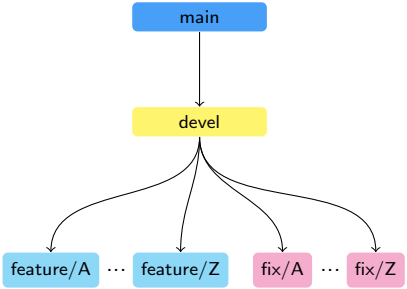
- There are various source control systems, but **Git is the most widespread system**, particularly due to platforms such as GitHub and GitLab, which significantly simplify the interaction with and management of Git repositories
- Git is a **decentralised source control system** — every developer has **their own copy of a repository** and changes are integrated into a main repository through **merge operations**
  - This is opposed to centalised source control systems, where commits are directly added to a central repository
- Due to its decentralised nature, Git **enables offline working**
  - In principle, Git can be used without a central repository — useful for maintaining local backups

Branches

- Collaborative development with repositories is typically done through **branches**

Branches

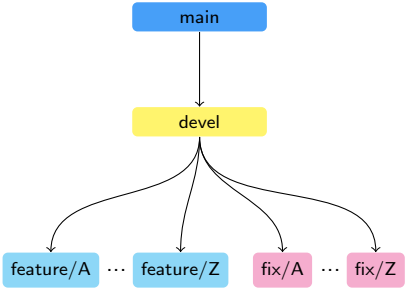
- Collaborative development with repositories is typically done through **branches**



- There are many possible **branching models** — each team may have its own conventions for that — but the following decomposition is one that is followed often:

Branches

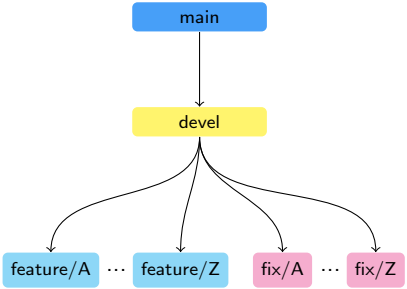
- Collaborative development with repositories is typically done through **branches**



- There are many possible **branching models** — each team may have its own conventions for that — but the following decomposition is one that is followed often:
  - A `main` branch that has a **stable version** of the code and which is **updated only occasionally** (e.g. twice a year)

Branches

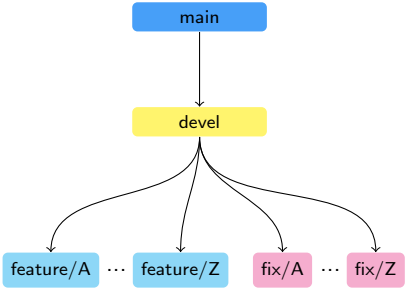
- Collaborative development with repositories is typically done through **branches**



- There are many possible **branching models** — each team may have its own conventions for that — but the following decomposition is one that is followed often:
  - A `main` branch that has a **stable version** of the code and which is **updated only occasionally** (e.g. twice a year)
  - A `devel` branch, which **combines new features and bug fixes** that are **planned to be integrated into `main`**

Branches

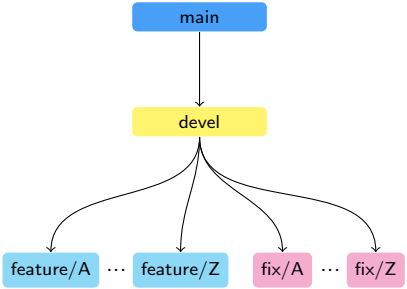
- Collaborative development with repositories is typically done through **branches**



- There are many possible **branching models** — each team may have its own conventions for that — but the following decomposition is one that is followed often:
  - A `main` branch that has a **stable version** of the code and which is **updated only occasionally** (e.g. twice a year)
  - A `devel` branch, which **combines new features and bug fixes** that are **planned to be integrated into `main`**
  - An arbitrary number of **feature** branches, each of which contains work-in-progress code for a new feature

Branches

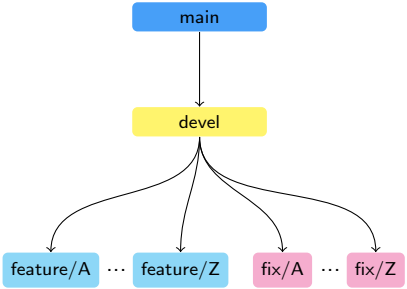
- Collaborative development with repositories is typically done through **branches**



- There are many possible **branching models** — each team may have its own conventions for that — but the following decomposition is one that is followed often:
  - A `main` branch that has a **stable version** of the code and which is **updated only occasionally** (e.g. twice a year)
  - A `devel` branch, which **combines new features and bug fixes** that are **planned to be integrated into `main`**
  - An arbitrary number of **feature** branches, each of which contains work-in-progress code for a new feature
  - An arbitrary number of **hotfix** branches, each containing work-in-progress code for a bug fix

Branches

- Collaborative development with repositories is typically done through **branches**



- There are many possible **branching models** — each team may have its own conventions for that — but the following decomposition is one that is followed often:
  - A `main` branch that has a **stable version** of the code and which is **updated only occasionally** (e.g. twice a year)
  - A `devel` branch, which **combines new features and bug fixes** that are **planned to be integrated into `main`**
  - An arbitrary number of **feature** branches, each of which contains work-in-progress code for a new feature
  - An arbitrary number of **hotfix** branches, each containing work-in-progress code for a bug fix

- Code from a child branch is integrated into a parent branch via a **merge (aka pull) request**

Forks

- Decentralised version control systems enable creating (independent) repository copies, called **forks**

Forks

- Decentralised version control systems enable creating (independent) repository copies, called **forks**
- A fork is **a personal, online copy of an existing repository**
  - Changes made in the forked repository are not reflected in the upstream repository — a fork thus allows to customise the original code based on one's own needs
  - But pull requests can also be made to the upstream repository

Forks

- Decentralised version control systems enable creating (independent) repository copies, called **forks**
- A fork is **a personal, online copy of an existing repository**
  - Changes made in the forked repository are not reflected in the upstream repository — a fork thus allows to customise the original code based on one's own needs
  - But pull requests can also be made to the upstream repository
- Forks **simplify repository access management for large, very distributed teams**
  - Instead of giving every contributor access to the main repository, everyone maintains their own fork and makes a pull request later
  - For this reason, forks are essential for large open-source projects, where anyone can contribute

Typical Collaborative Development Workflow

- Code on branches and/or forks is integrated via **pull requests**

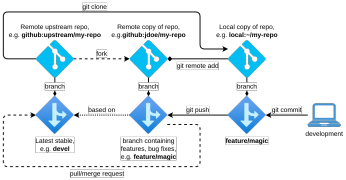


Illustration of a typical collaborative development workflow. Courtesy of Minh Nguyen.

Typical Collaborative Development Workflow

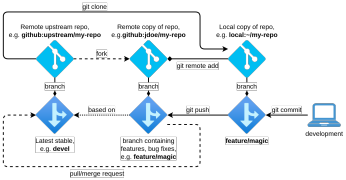


Illustration of a typical collaborative development workflow. Courtesy of Minh Nguyen.

- Code on branches and/or forks is integrated via **pull requests**
- A pull request is always created **against a specific branch** — if accepted, the changes will be merged in that branch
  - The branch that contains changes is usually referred to as a **source branch**; the branch in which the changes should be merged is called a **target branch**

Typical Collaborative Development Workflow

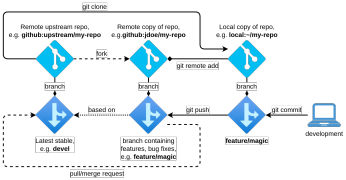


Illustration of a typical collaborative development workflow. Courtesy of Minh Nguyen.

- Code on branches and/or forks is integrated via **pull requests**
- A pull request is always created **against a specific branch** — if accepted, the changes will be merged in that branch
  - The branch that contains changes is usually referred to as a **source branch**; the branch in which the changes should be merged is called a **target branch**
- A typical workflow for collaborative development goes through the following steps:

Typical Collaborative Development Workflow

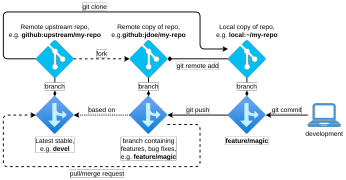


Illustration of a typical collaborative development workflow. Courtesy of Minh Nguyen.

- Code on branches and/or forks is integrated via **pull requests**
- A pull request is always created **against a specific branch** — if accepted, the changes will be merged in that branch
  - The branch that contains changes is usually referred to as a **source branch**; the branch in which the changes should be merged is called a **target branch**
- A typical workflow for collaborative development goes through the following steps:
  1. **Changes are made locally and are pushed to a branch**, potentially in a personal fork

Typical Collaborative Development Workflow

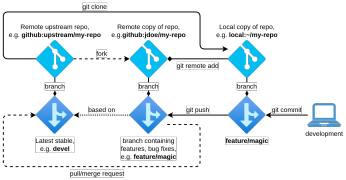


Illustration of a typical collaborative development workflow. Courtesy of Minh Nguyen.

- Code on branches and/or forks is integrated via **pull requests**
- A pull request is always created **against a specific branch** — if accepted, the changes will be merged in that branch
  - The branch that contains changes is usually referred to as a **source branch**; the branch in which the changes should be merged is called a **target branch**
- A typical workflow for collaborative development goes through the following steps:
  1. **Changes are made locally and are pushed to a branch**, potentially in a personal fork
  2. Once the development of the feature or fix is completed, **a pull request is created**

Typical Collaborative Development Workflow

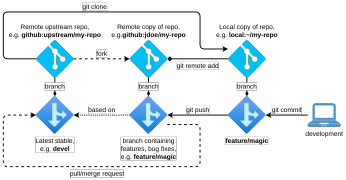


Illustration of a typical collaborative development workflow. Courtesy of Minh Nguyen.

- Code on branches and/or forks is integrated via **pull requests**
- A pull request is always created **against a specific branch** — if accepted, the changes will be merged in that branch
  - The branch that contains changes is usually referred to as a **source branch**; the branch in which the changes should be merged is called a **target branch**
- A typical workflow for collaborative development goes through the following steps:
  1. **Changes are made locally and are pushed to a branch**, potentially in a personal fork
  2. Once the development of the feature or fix is completed, **a pull request is created**
  3. **Feature and fix branches are usually deleted after being merged**

Merge Conflicts

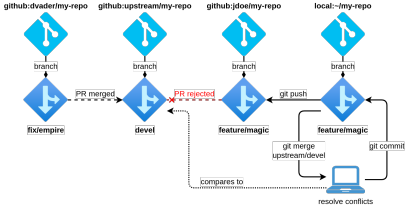


Illustration of a pull request workflow where merge conflicts appear.
Courtesy of Minh Nguyen.

- When a pull request is made, it may happen that there is a **merge conflict** that makes it impossible to merge the changes directly
  - In this case, **conflict resolution** is necessary before a merge is possible

Merge Conflicts

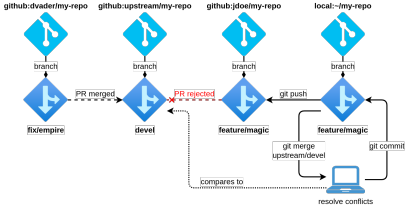


Illustration of a pull request workflow where merge conflicts appear.
Courtesy of Minh Nguyen.

- When a pull request is made, it may happen that there is a **merge conflict** that makes it impossible to merge the changes directly
  - In this case, **conflict resolution** is necessary before a merge is possible
- A merge conflict appears when **one or more files that are included in the pull request have been edited on the target branch** since the last time the source branch was synchronised with its parent

Merge Conflicts

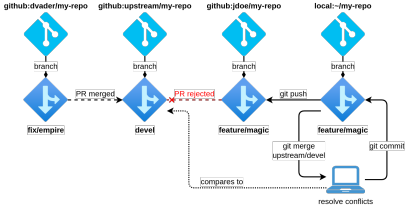


Illustration of a pull request workflow where merge conflicts appear. Courtesy of Minh Nguyen.

- When a pull request is made, it may happen that there is a **merge conflict** that makes it impossible to merge the changes directly
  - In this case, **conflict resolution** is necessary before a merge is possible
- A merge conflict appears when **one or more files that are included in the pull request have been edited on the target branch** since the last time the source branch was synchronised with its parent
- Conflict resolution is a **manual process** that requires **careful consideration of the conflicting file lines** so that important changes are not lost

Merge Conflicts

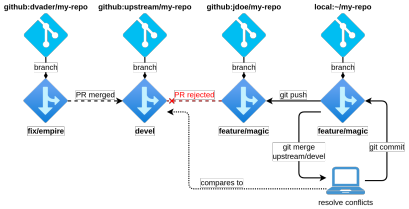
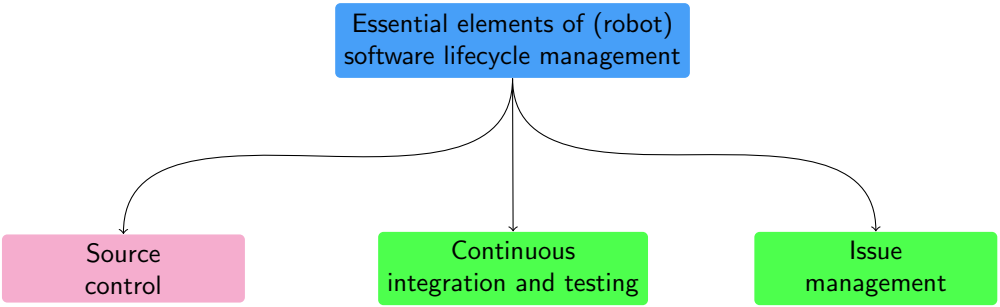


Illustration of a pull request workflow where merge conflicts appear. Courtesy of Minh Nguyen.

- When a pull request is made, it may happen that there is a **merge conflict** that makes it impossible to merge the changes directly
  - In this case, **conflict resolution** is necessary before a merge is possible
- A merge conflict appears when **one or more files that are included in the pull request have been edited on the target branch** since the last time the source branch was synchronised with its parent
- Conflict resolution is a **manual process** that requires **careful consideration of the conflicting file lines** so that important changes are not lost
- The best strategy to avoid merge conflicts is to **pull from the parent branch / repository frequently** so that new file edits are integrated before a pull request is even created


Essential elements of (robot) software lifecycle management
Source control
Continuous integration and testing
Issue management

Unit Testing

- As discussed in our first lecture of the course, **testing is one of the essential elements of (robot) software development**

Unit Testing

- As discussed in our first lecture of the course, **testing is one of the essential elements of (robot) software development**

- A robot has many components though, such that we are interested in whether they all work correctly as individual units before they can be integrated together

Unit Testing

- As discussed in our first lecture of the course, **testing is one of the essential elements of (robot) software development**
- A robot has many components though, such that we are interested in whether they all work correctly as individual units before they can be integrated together
- Unit testing is **a process of testing the correctness of individual components** in a code base

Unit Testing

- As discussed in our first lecture of the course, **testing is one of the essential elements of (robot) software development**
- A robot has many components though, such that we are interested in whether they all work correctly as individual units before they can be integrated together
- Unit testing is **a process of testing the correctness of individual components** in a code base
- Most languages have standard libraries for writing and executing unit tests, e.g. **unittest** in Python

Unit Testing

- As discussed in our first lecture of the course, **testing is one of the essential elements of (robot) software development**
- A robot has many components though, such that we are interested in whether they all work correctly as individual units before they can be integrated together
- Unit testing is **a process of testing the correctness of individual components** in a code base
- Most languages have standard libraries for writing and executing unit tests, e.g. **unittest** in Python

Example of a Python-based unit test for a library that interacts with data in a database. Taken from https://github.com/ropod-project/black-box-tools

```python
class TestDataUtils(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.test_db_name = "bb_tools_test_data"
        test_dir = os.path.abspath(os.path.dirname(__file__))
        cls.test_db_dir = os.path.join(test_dir, cls.test_db_name)
        cls.collection_name = 'ros_ropod_cmd_vel'

        host, port = cls._get_db_host_and_port()
        cls.client = pm.MongoClient(host=host, port=port)
        success = cls._restore_test_db()
        assert(success)

    @classmethod
    def tearDownClass(cls):
        cls._drop_test_db()

    def test_get_all_measurement(self):
        database = self.client[self.test_db_name]
        collection = database[self.collection_name]
        doc_cursor = collection.find({})
        docs = [doc for doc in doc_cursor]
        measurements = DataUtils.get_all_measurements(docs, 'linear/x')
        self.assertEqual(measurements.shape, (149,))
```

Continuous Integration and Unit Test Automation

- Source control is typically done in conjunction with **automated tools** that simplify the integration and deployment process

Continuous Integration and Unit Test Automation

- Source control is typically done in conjunction with **automated tools** that simplify the integration and deployment process

An example of a continuous integration pipeline on GitHub. Taken from https://github.com/ropod-project/black-box-tools

```
image: "mongo"
before_script:
    - apt-get update
    - apt-get install -y git-all
    - apt-get install -y python3-pip
    - pip3 install --upgrade pip
    - cat requirements.txt | xargs -n 1 -L 1 pip3 install
variables:
  DB_HOST: mongo
services:
  - mongo
test:
    script:
        - python3 -m unittest discover -s 'test' -p 'test_*.py'
```

- **Continuous integration** is one such automated process, based on which changes are merged into a target branch at a frequent rate — in principle, multiple times a day

Continuous Integration and Unit Test Automation

- Source control is typically done in conjunction with **automated tools** that simplify the integration and deployment process

An example of a continuous integration pipeline on GitHub. Taken from https://github.com/ropod-project/black-box-tools

```
image: "mongo"
before_script:
    - apt-get update
    - apt-get install -y git-all
    - apt-get install -y python3-pip
    - pip3 install --upgrade pip
    - cat requirements.txt | xargs -n 1 -L 1 pip3 install
variables:
  DB_HOST: mongo
services:
  - mongo
test:
    script:
        - python3 -m unittest discover -s 'test' -p 'test_*.py'
```

- **Continuous integration** is one such automated process, based on which changes are merged into a target branch at a frequent rate — in principle, multiple times a day
- **Continuous integration pipelines are typically automated** and are associated with **automated tests** — the code changes are integrated into the target branch **only if all tests pass**

Continuous Integration and Unit Test Automation

- Source control is typically done in conjunction with **automated tools** that simplify the integration and deployment process

An example of a continuous integration pipeline on GitHub. Taken from https://github.com/ropod-project/black-box-tools

```
image: "mongo"
before_script:
    - apt-get update
    - apt-get install -y git-all
    - apt-get install -y python3-pip
    - pip3 install --upgrade pip
    - cat requirements.txt | xargs -n 1 -L 1 pip3 install
variables:
  DB_HOST: mongo
services:
  - mongo
test:
    script:
        - python3 -m unittest discover -s 'test' -p 'test_*.py'
```

- **Continuous integration** is one such automated process, based on which changes are merged into a target branch at a frequent rate — in principle, multiple times a day
- **Continuous integration pipelines are typically automated** and are associated with **automated tests** — the code changes are integrated into the target branch **only if all tests pass**
- Additional steps, such as **static code analysis** tools, are often included in continuous integration pipelines as well — this **prevents low-quality code from being merged**

Hardware-in-the-Loop (HIL) Testing

- Robots are embodied agents, so **pure software testing of a robot is never sufficient** — real testing has to be performed eventually

Hardware-in-the-Loop (HIL) Testing

- Robots are embodied agents, so **pure software testing of a robot is never sufficient** — real testing has to be performed eventually
- HIL testing is a procedure based on which **real signals are used to test the system in a simulated environment**
  - This simplifies the process of testing with the real system since there is no need to expose it to potentially hazardous situations

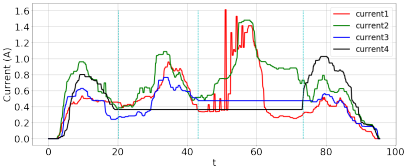
Hardware-in-the-Loop (HIL) Testing

- Robots are embodied agents, so **pure software testing of a robot is never sufficient** — real testing has to be performed eventually
- HIL testing is a procedure based on which **real signals are used to test the system in a simulated environment**
  - This simplifies the process of testing with the real system since there is no need to expose it to potentially hazardous situations
- **The quality of HIL testing depends on the used simulation though** — a high-fidelity simulation is needed so that meaningful evaluation results are obtained

Hardware-in-the-Loop (HIL) Testing

- Robots are embodied agents, so **pure software testing of a robot is never sufficient** — real testing has to be performed eventually
- HIL testing is a procedure based on which **real signals are used to test the system in a simulated environment**
  - This simplifies the process of testing with the real system since there is no need to expose it to potentially hazardous situations
- **The quality of HIL testing depends on the used simulation though** — a high-fidelity simulation is needed so that meaningful evaluation results are obtained
- **Testing with the real system under real conditions is eventually unavoidable**, but it is best if this is only performed after software-based and HIL testing are both satisfactory

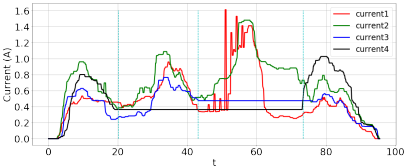
Fault Injection



Current measurements of a Robile-like platform with injected wheel faults. Taken from A. Mitrevski and P. G. Plöger, "Data-Driven Robot Fault Detection and Diagnosis Using Generative Models: A Modified SFDD Algorithm," in *30th Int. Workshop Principles of Diagnosis (DX)*, 2019.

- Robots process complex sensor data to eventually produce actuator output, but **sensors, actuators, as well as algorithms are all susceptible to faults**
  - Faults can be either **intermittent**, which appear sporadically, or **permanent**, which persist for a prolonged period of time

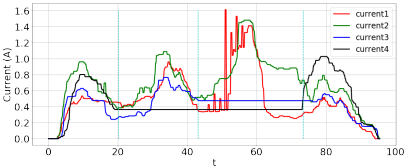
Fault Injection



Current measurements of a Robile-like platform with injected wheel faults. Taken from A. Mitrevski and P. G. Plöger, "Data-Driven Robot Fault Detection and Diagnosis Using Generative Models: A Modified SFDD Algorithm," in *30th Int. Workshop Principles of Diagnosis (DX)*, 2019.

- Robots process complex sensor data to eventually produce actuator output, but **sensors, actuators, as well as algorithms are all susceptible to faults**
  - Faults can be either **intermittent**, which appear sporadically, or **permanent**, which persist for a prolonged period of time
- It is thus usually of interest to know **how a robot reacts under noisy or faulty condition**

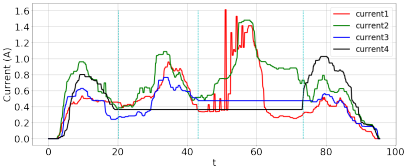
Fault Injection



Current measurements of a Robile-like platform with injected wheel faults. Taken from A. Mitrevski and P. G. Plöger, "Data-Driven Robot Fault Detection and Diagnosis Using Generative Models: A Modified SFDD Algorithm," in *30th Int. Workshop Principles of Diagnosis (DX)*, 2019.

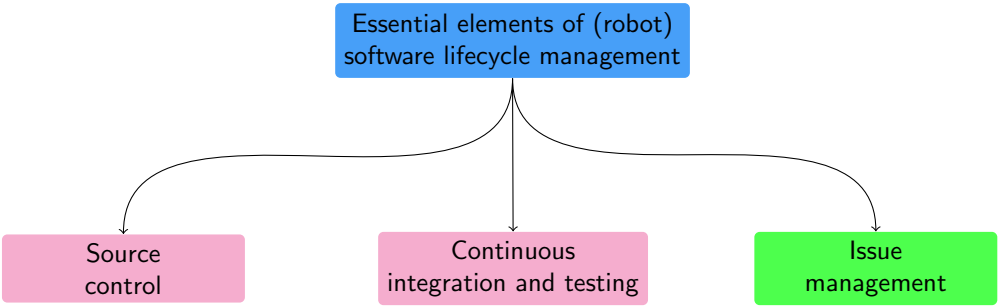
- Robots process complex sensor data to eventually produce actuator output, but **sensors, actuators, as well as algorithms are all susceptible to faults**
  - Faults can be either **intermittent**, which appear sporadically, or **permanent**, which persist for a prolonged period of time
- It is thus usually of interest to know **how a robot reacts under noisy or faulty condition**
- **Fault injection** is a procedure based on which **faults are induced at particular places in the software** so that the **system's resilience to failures** can be examined

Fault Injection



Current measurements of a Robile-like platform with injected wheel faults. Taken from A. Mitrevski and P. G. Plöger, "Data-Driven Robot Fault Detection and Diagnosis Using Generative Models: A Modified SFDD Algorithm," in *30th Int. Workshop Principles of Diagnosis (DX)*, 2019.

- Robots process complex sensor data to eventually produce actuator output, but **sensors, actuators, as well as algorithms are all susceptible to faults**
  - Faults can be either **intermittent**, which appear sporadically, or **permanent**, which persist for a prolonged period of time
- It is thus usually of interest to know **how a robot reacts under noisy or faulty condition**
- **Fault injection** is a procedure based on which **faults are induced at particular places in the software** so that the **system's resilience to failures** can be examined
- Fault injection can also be used to **verify the correctness of fault management strategies**, namely fault detection, identification, and diagnosis

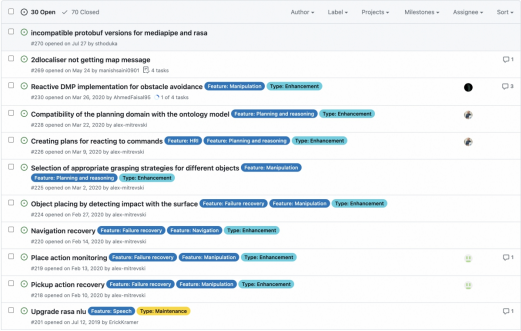

Essential elements of (robot) software lifecycle management
Source control
Continuous integration and testing
Issue management

Task Management Through Issues

- When working on a complex system such as a robot, there are typically **many tasks that need to be performed at any point in time** — this creates the need for task management

Task Management Through Issues

- When working on a complex system such as a robot, there are typically **many tasks that need to be performed at any point in time** — this creates the need for task management

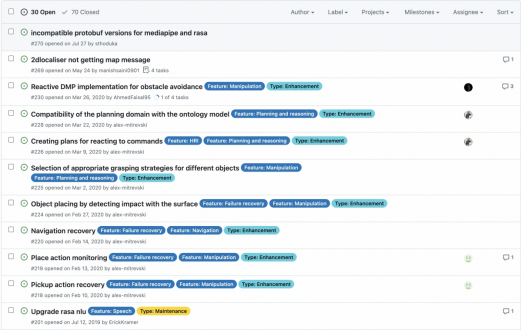


A list of open issues from https://github.com/b-it-bots/mas_domestic_robotics/issues

- Task management and allocation is often performed **via issues**

Task Management Through Issues

- When working on a complex system such as a robot, there are typically **many tasks that need to be performed at any point in time** — this creates the need for task management

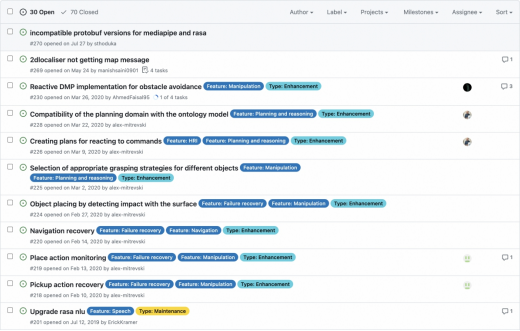


A list of open issues from https://github.com/b-it-bots/mas_domestic_robotics/issues

- Task management and allocation is often performed **via issues**
- An issue is a **description of a concrete task** that can be **assigned to a developer**, potentially with **associated tags** that help identify the type of issue

Task Management Through Issues

- When working on a complex system such as a robot, there are typically **many tasks that need to be performed at any point in time** — this creates the need for task management



A list of open issues from https://github.com/b-it-bots/mas_domestic_robotics/issues

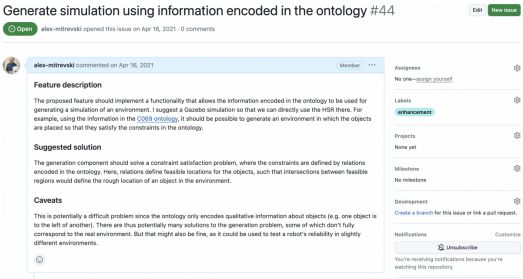
- Task management and allocation is often performed **via issues**
- An issue is a **description of a concrete task** that can be **assigned to a developer**, potentially with **associated tags** that help identify the type of issue
- Issues are typically **associated with individual repositories**, but can also be managed for a complete project — this depends on the development style that is followed by a team

Issue Best Practices

- Issues should **describe a problem as precisely** and should be **as concrete as possible**
  - Although it may sometimes be desirable to organise issues hierarchically

Issue Best Practices

- Issues should **describe a problem as precisely** and should be **as concrete as possible**
  - Although it may sometimes be desirable to organise issues hierarchically

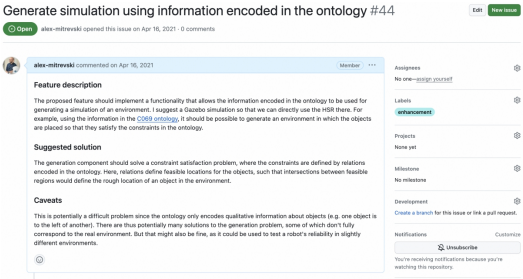


An example issue describing a new feature (taken from https://github.com/b-it-bots/mas_knowledge_base)

- Issue descriptions differ based on **the purpose for which they are created**:

Issue Best Practices

- Issues should **describe a problem as precisely** and should be **as concrete as possible**
  - Although it may sometimes be desirable to organise issues hierarchically

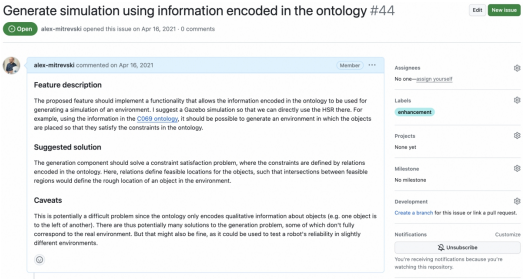


An example issue describing a new feature (taken from https://github.com/b-it-bots/mas_knowledge_base)

- Issue descriptions differ based on **the purpose for which they are created**:
  - Feature requests **describe a feature that is missing and that would be good to add**

Issue Best Practices

- Issues should **describe a problem as precisely** and should be **as concrete as possible**
  - Although it may sometimes be desirable to organise issues hierarchically

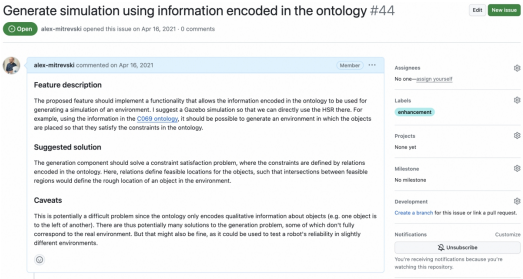


An example issue describing a new feature (taken from https://github.com/b-it-bots/mas_knowledge_base)

- Issue descriptions differ based on **the purpose for which they are created**:
  - Feature requests **describe a feature that is missing and that would be good to add**
  - Bug reports **describe an observed problem with the software**, such that they should include **the steps to reproduce the bug** and typically **a description of the hardware and software configuration under which the bug was observed**

Issue Best Practices

- Issues should **describe a problem as precisely** and should be **as concrete as possible**
  - Although it may sometimes be desirable to organise issues hierarchically

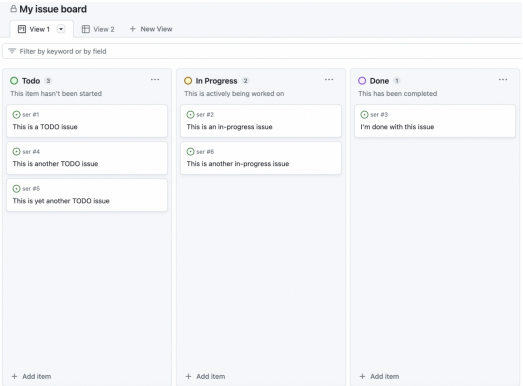


An example issue describing a new feature (taken from https://github.com/b-it-bots/mas_knowledge_base)

- Issue descriptions differ based on **the purpose for which they are created**:
  - Feature requests **describe a feature that is missing and that would be good to add**
  - Bug reports **describe an observed problem with the software**, such that they should include **the steps to reproduce the bug** and typically **a description of the hardware and software configuration under which the bug was observed**

- Many open-source repositories have **issue description templates** — the details expected in the template should all be filled out for the issue to be considered by the developers

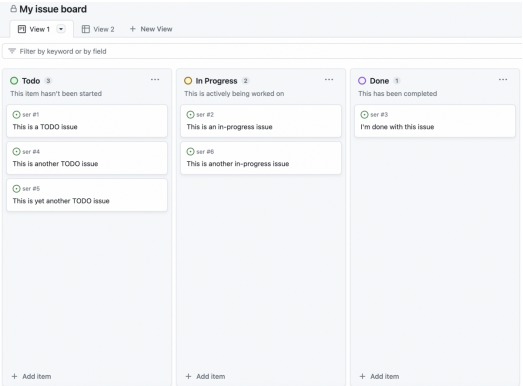
Issue Boards



An example of an issue board on GitHub

- **Particularly in large(r) projects, issue management can become a daunting task** — if there are many issues in a repository, it can become challenging to track which issues are open, in progress, done, or even abandoned

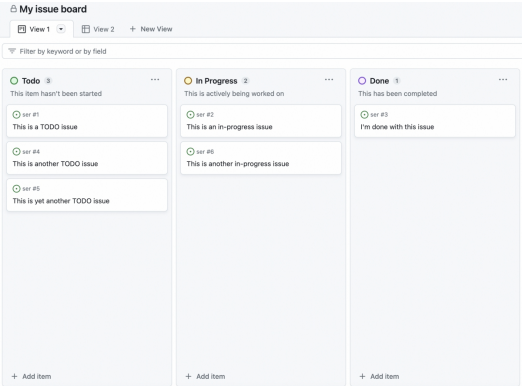
Issue Boards



An example of an issue board on GitHub

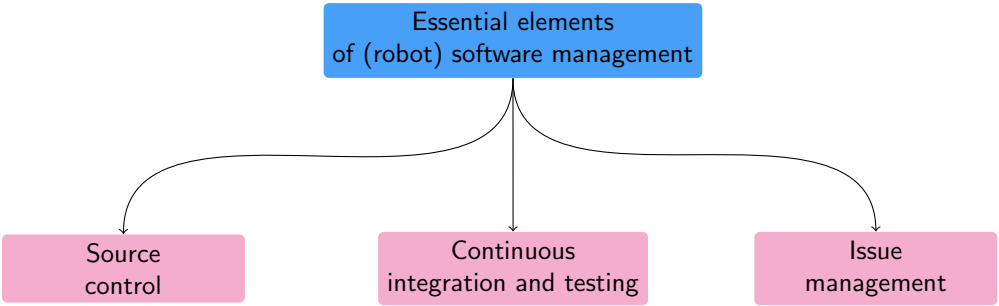
- **Particularly in large(r) projects, issue management can become a daunting task** — if there are many issues in a repository, it can become challenging to track which issues are open, in progress, done, or even abandoned
- **Issue boards** can simplify this process by **organising issues into different columns**, such as `open`, `in progress`, and `completed`
  - An issue board is a visual representation of the product backlog in scrum, which we discussed a few weeks ago

Issue Boards



An example of an issue board on GitHub

- **Particularly in large(r) projects, issue management can become a daunting task** — if there are many issues in a repository, it can become challenging to track which issues are open, in progress, done, or even abandoned
- **Issue boards** can simplify this process by **organising issues into different columns**, such as `open`, `in progress`, and `completed`
  - An issue board is a visual representation of the product backlog in scrum, which we discussed a few weeks ago
- Most major software management platforms, such as GitHub and GitLab, provide an integrated facility for creating issue boards


Essential elements
of (robot) software management
Source
control
Continuous
integration and testing
Issue
management