Anu Ramanathan

anur\_tw@outlook.com

STEPS TO CODE SHARING & MULTI-TENANCY

An Analysis of Steps Involved in Updating Development Team Workflow

Abstract

This holds the results of an analysis of changes that are necessary to upgrade current developer workflow from a single active developer using their own method to store code updates and with no file versioning system. The aim is to change to a distributed, remote-first team with a streamlined workflow against an open-source code repository. The process to implement Continuous Integration and Continuous Deployment (CI/CD) with its attendant automation is another step forward. The choice of a code-repository and a cloud-native multi-tenant service, therefore, becomes critical.

Table of Contents

[Online Code Repository](#_Online_Code_Repository) 2

[Data Management](#_Data_Management) 2

[Pre-requisites for Continuous Integration/Continuous Deployment](#_Pre-requisites_for_Continuous) 3

[Setting up a Cloud Administrative Toolchain](#_Setting_up_a) 5

[Setting up CI/CD](#_Setting_up_CI/CD) 6

Appendix A: [Repository Listing and Comparison](#_Appendix_A:_Repository) 7

Appendix B: [Dictionary of Terms](#_Appendix_B:_Dictionary) 9

Steps to Code Sharing & Multi-Tenancy

# Online Code Repository

1. Selecting a **code repository** to work with- see [**Appendix A**](#_Appendix_A) for a table listing the pros and cons of each tool. Below, there is a list the most useful choices (in order of appearance) to fulfil the listed goals of [*redacted*].

**Git** *and* **Mercurial** *are the most popular distributed source code repositories.*

* 1. **GitLab**, which implements **Git**, is a DevOps platform with integrated **Continuous Integration & Continuous Deployment** (CI/CD) in its toolchain.
  2. **Fossil** and **BitBucket** both implement **Mercurial** as their distributed, open-source code repository.
  3. **GitHub**, also implementing **Git**, is much like **GitLab**.
  4. For version control, **TFS** + **Azure DevOps** offers a choice of **Team Foundation Version Control** (**TFVC**) with its centralized architecture or **Git** with its distributed architecture.
  5. **SVN** (Subversion) has a decentralized architecture and works better in situations with co-located teams.
  6. **Qualities required** in the code repository are:
     1. Does it allow **branching**?
     2. Does it have an **active user community**?
     3. Does *redacted* product have to have any form of **license statement in its code base**?
     4. Does the repository include **bug/task tracking** and **integrate easily with external tools** like Jira, Trello, et cetera?
     5. Does the repository make your code **accessible online** as easily as on your computer?
     6. Does the repository support **DevOps** and **DevSecOps**?
     7. Does the repository make it easy to implement **CI/CD** with easily available tools? In my experience, most command line tools in open source tend to be free, including those for CI/CD.
  7. **Nice-to-have qualities** in the code repository include:
     1. Is it an **open-source** tool? **Task:** Add notes here to explain why having an open-source tool is good to have.

# Data Management

1. Moving all aspects of **data management** into MySQL:
   1. [*redacted*] have had plans to **move all the data** into **MySQL** and will have moved forward in this effort since early 2020 when I last interacted with them.
      1. Based on my knowledge of the situation, [*redacted*] have successfully moved **only** **reporting data** into a MySQL database, hosted on an internal server. Reporting data lags transactional data by a day or so.
      2. The long-term plan was to **use the transaction data store** itself as the **source of truth** for up-to-the-minute **reporting**.
      3. Depending on the frequency of reporting and size of the transaction database, this decision is something that [*redacted*] will have to revisit after the rest of the process has been set up.
   2. [*Redacted*] has been creating a **ReSTful Service** with end points for all four basic data operations viz. **C**reate/add, **R**ead, **U**pdate, and **D**elete (also known as **CRUD**) for all the tables in its database, to have only **one source of truth** between the standalone application and the planned web application and web site rebuild.
      1. As part of the ReSTful Service build, [*redacted*] has moved to using JavaScript Object Notation (**JSON**) as the preferred data transfer format.
   3. The MySQL instances will move **out of on-premises installation** on company-owned or client-owned hardware **into horizontally and vertically scalable** online instances, using any one of the cloud service providers- AWS, GCP, Azure, among others- depending on cost per unit capacity, among other considerations.

# Pre-requisites for Continuous Integration/Continuous Deployment

1. **Pre-requisites** for CI/CD:
   1. Set up a **test plan** for the full product such that the test suite only runs for the **actual installation** for a customer:
      1. The variable components of the product configuration functional test suite or client scenario test suite is the only thing that is **different** **between clients** and is part of their product setup or upgrade.
      2. The installation technician **configures the test suite** according to the test plan. The options are to cover only the core product with necessary components, or cover both the core product and the optional parts that the client has chosen to buy or upgrade to.
      3. The technician executes the configured test plan for the client by running the test suite configured for the product installation for the client, especially the **client-specific scenario tests**.
   2. Create a **robust test suite** for the product, so that the current action does not move forward unless the technician can show a *“clean”* test run.
      1. Start with **full-coverage unit tests**, usually written by the developers themselves, especially when using a system like DevOps whose workflow encourages it.
      2. Roll up the unit tests into **module tests**, which should also assess for any **data dependencies**.
      3. Unit tests, no matter how comprehensive, should be only part of the tests for each module. **Test developers** (also called “Software Design Engineers in Test” or SDETs) add **module-specific tests**.
      4. Roll up the module tests into **integration tests**. The same process for module tests applies to the integration tests too. Here the **test developer** takes the leadagain.
      5. Collaborating with the rest of the team improves the accuracy, speed, and impact of the integration tests.
      6. This is especially important for the optional modules or those that have dependencies of any kind. However, good coding practice is to not allow direct cross-module dependencies whether in code, data structures, or data flows.
      7. Roll up the integration tests into **full product tests**, which include each combination of the core product with its required modules and each set of optional module configurations that [*redacted*] offer.
      8. Create **client scenario tests** against the product depending on each product configuration. Gather them into a **client scenario test suite** (also known as a **product configuration functional test suite**).

# Setting up a Cloud Administrative Toolchain

1. Setting up the Cloud Administrative Toolchain consists of the following steps:
   1. Open a business account with one of the Cloud capacity providers, of which the following three are most popular:
      1. AWS (by Amazon)
      2. GCP (by Google)
      3. Azure (by Microsoft)
   2. **Create automated jobs** that can stand up and tear down virtual machines to conform to the specified environment desired.
   3. Create other automated jobs that can set up the required dev, test, and production environments, **starting with a clean state each time**.
   4. The administrator creates automated jobs and tests that virtual systems can be added and removed for **each** of the following **situations** (see [Appendix B](#_Appendix_B:_Dictionary) for why virtual systems are the favored option):
      1. **Standard developer/tester system**, fully configured with all development, build, and test tools in the toolchain, and ready for use by developers and testers. These systems are normally set up with more memory, disk space, and faster processers to allow developers and testers reach maximum productivity.
      2. **Standard technical writer system**, fully configured with all documentation tools needed, and ready for use. These systems may be just a standard office user system with documentation tools and more disk space.
      3. **Standard general office user system** with no developer- or writer-specific tools installed, fully configured and ready for use by team members in all other roles.
      4. **A build system**, minimally configured with only the tools used within the CI/CD toolchain, to run the check-in tests to allow developers to save their updates to the selected code repository and executing all post-build tasks that include everything up to product packaging for release.
      5. **White-box test system** set up to run the **full product test suite**, and the **client scenario test suite** with the assumption that all optional modules form part of the offered product configuration. This is for **pre-release testing**.
      6. **White-box test systems** set up for the **core product** and each **combination of optional modules** that is available to a client, able to run the **product-** and **functional test suites**. There are multiple such systems needed to test each of the configurations that the product can support.
      7. ***Black-box* user acceptance test (UAT) systems** that the system creation test script sets up to run test suites for only the core product and optional modules a client has chosen. Then run the **client-specific functional test suite** to be sure that the system functions as expected by the client.
      8. **Post-release black-box test systems** set up for the latest released build of the core product and all modules properly configured. These run only the **functional test suite** and that the test developer changes to match each official product release.
   5. Build up a **test library** that can run in the following **situations**:
      1. **Ad-hoc test runs** against a development or release branch of the code base. In this case, **the full product test suite** runs against the entire code base.
      2. At **each code check-in**, where the check-in cannot to go forward until a successful **full product test suite** run is successful. This also functions as a kind of regression test to catch any unintentional dependencies across the product.
      3. If anyone (including the development team) **reports a product issue**, configure a spin-up instance corresponding to the installed system of the reporter for thorough testing and debugging. The aim should be for this to be rare.
   6. Use a **robust ticketing system** to keep track of all help desk calls (which technician picked up the ticket, what the solution was, notes, and comments), reported issues, and bugs. Product improvement suggestions also go here. Aggregating this to show in the dashboard for the product manager will be invaluable to them.

# Setting up CI/CD

1. Assuming [*redacted*] has already selected an online code repository and that the product code is there, set up a **DevOps pipeline** that enforces the following:
   1. No code check-ins allowed unless ***all*** the following are **true**:
      1. **Code reviews** are complete, and the developer has implemented the requested changes.
      2. The complete automated product test suite **can run without errors** using build scripts that pull in the latest checked in code within the **current** (normally Development) branch of the code repository.
      3. Each check-in has a **comprehensive check-in comment** that lists the specific **product development** **task**, **bug report**, and a **summary of the changes**.
   2. The automated build script rejects any check-in without links back to a specific task or bug report or missing a summary of changes. Otherwise, the check-in is accepted, and the change is moved into the code repository.
   3. A successful check-in **triggers the creation of a virtual system** with the core product and all modules installed and configured.
   4. The build script then automatically runs the **full functional test suite**, which should run clean (without errors) before moving the checked-in code changes to a **release branch** within the code repository.

# Appendix A: Repository Listing and Comparison

| Repository | Branching Support | Open Source | Major Features | License Type | Issues/ Task Tracking | Code accessibility | CI/CD Support | Active User Community | Max Price Per Month (Top License) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| GitLab | Supported | Open Source | Distributed Architecture,  DevOps platform | Ultimate | Enabled | Online, Desktop | Integrated | Active | US$ 99 per user per month |
| Fossil | Supported | Open Source | Distributed Architecture, Wiki, Forum, Email Alerts, Chat, Autosync, Self-Contained Executable, Git Mirroring, can be set up with Mercurial Code Repository | 2-clause BSL | Enabled | Desktop | Integrated | Reasonably Active | Free |
| Atlassian BitBucket | Supported | Open Source | Based on Git, Distributed Architecture,  DevSecOps, Tool Integrations with Jira, Trello | Premium | Yes | Online, Desktop | Integrated | Reasonably Active | US$ 6 per user per month |
| Atlassian BitBucket Data Center | Supported | Open Source | Based on Git, Distributed Architecture,  DevSecOps | Enterprise | Yes | Online, Desktop | Integrated | Inactive | US$ 200 per month  US$ 2,300 per year, self-hosted |
| Mercurial | Supported | Open Source | Distributed Architecture,  Platform Independent | Gnu GPLv2 | Yes, by linking to other services | Online, Desktop | Possible with external toolchain | Active | Free |
| Git/GitGui | Supported | Open Source | Distributed Architecture,  DevOps | Gnu GPLv2 | Enabled | Online,  Desktop | Supported | Active | Free |
| Microsoft GitHub | Supported | Closed Source | Distributed Architecture,  DevOps available, SAML SSO | Enterprise | Enabled | Online, Desktop | Integrated | Active | US$ 315 per month + extras |
| Apache Subversion | Not a built-in concept | Open Source | No DevOps,  Centralized Architecture | Apache-2.0 | Unknown | Desktop | Not supported | Active | Free |

# 

# Appendix B: Dictionary of Terms

| **#** | **Term or Phrase** | **Explanation** | **Context** |
| --- | --- | --- | --- |
|  | Revision Control System | [\*](#SubText1)From [Wikipedia](https://en.wikipedia.org/wiki/Version_control)[[1]](#RefSuperText1)  In software engineering, any kind of practice that tracks and gives users control over changes to source code. Sometimes developers and writers use revision control software to write documentation and keep it updated, where configuration files as well as source code are readily available. |  |
|  | Code Repository | [\*](#SubText1)From [Wikipedia](https://en.wikipedia.org/wiki/Repository_(version_control))[[2]](#RefSuperText2)  [In revision control systems, a code repository is a data structure that stores metadata for a set of files or directory structure. Depending on whether the version control system used is distributed or centralized, the complete set of information in the repository may be duplicated on every user’s system or may be kept on a single server.](#RefSuperText2)*[2.1]* The metadata in the repository includes:   * A historical record of changes in the repository, * A set of commit objects, * A set of references to commit objects, called heads. |  |
|  | DevOps | [\*](#SubText1)From [Wikipedia](https://en.wikipedia.org/wiki/DevOps)[[3]](#RefSuperText3)  DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. DevOps is complementary with Agile software development; several DevOps aspects came from the Agile method.  [\*](#SubText1)From NewRelic [[3.1]](#RefSuperText31)  DevOps stands for a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach.[*[3.2]*](#RefSuperText32) DevOps emphasizes people (and culture) and is a way to improve collaboration between operations and development teams. DevOps implementations use technology— especially automation tools, which can use an increasingly programmable and dynamic infrastructure from a life cycle perspective. [*[3.3]*](#RefSuperText33) | CI/CD |
|  | DevSecOps | DevSecOps is an improvement to DevOps, in that security is considered and included in the product from first conception all the way to product release.  [\*](#SubText1)From Wikipedia[[4]](#RefSuperText4) [[4.1]](#RefSuperText41)  DevSecOps is an augmentation of the DevOps workflow which allows security practices to be integrated into the DevOps approach. "The traditional centralized security team must adopt a federated model allowing each delivery team the ability to factor in the correct security controls into their DevOps practices."*[*[*4.2*](#RefSuperText42)*]* "Shifting security left is an approach to software security whereby security practices and testing are performed earlier in the development lifecycle." *[4.3]* |  |
|  | Remote-first Distributed Team | [\*](#SubText1)One or more members of remote and distributed teams work 100% remotely and may never come into a shared office at all. A maxim here is that ensuring remote access works to ensure equal in-office access too.  *Companies and teams can get ready for remote-first* [[5](#RefSuperText5)] *work by:*   * *Building new or changing existing workflows and communication paths to support equal access regardless of location.*    + Example issue*: Remote team members are unable to understand everything that is going on in the in-person meeting space and can miss key decisions, even when they have audio and video access to the meeting room. In-person attendees generally do not and because of that, they hold an information advantage over their remote colleagues.*   + Fix*: Change meeting spaces so that everyone calls in to online meetings from their desk. The problem of unequal access to team and work information is overcome because everyone in the meeting has the same access to resources like documents and key decisions.* * *Having real time meetings kept to a minimum and usually recorded for all.* * *Having decisions are made online, not offline at happy hours or at the water-cooler or coffee station.* * *Measuring performance by output rather than hours worked.* * *Instilling a culture of documentation where communication is asynchronous-first (for example, using Slack or other messaging systems).* * *Encouraging and empowering managers to work from home.* * *Ensuring everyone has an equal voice.* * *Awarding promotions more fairly based on a team member’s contributions and output, equally for remote vs. in-office team members. This is a unique way of working, a 180° twist away from how most companies have traditionally awarded promotions by rewarding those who show up to the office regardless of how well they can do their job.* * *Implementing radical transparency, never leaving a remote employee wondering if there is some information other team members have that is hidden from them.* |  |
|  | Continuous Integration and Continuous Deployment | [\*](#SubText1)Adapted from various [[6](#RefSuperText6)] online [[6.1](#RefSuperText61)] sources [[6.2]](#RefSuperText62)  CI and CD stand for continuous integration and continuous delivery/continuous deployment, a modern software development practice. Continuous Integration and Continuous Delivery (CI/CD) is a tangible implementation relying on automation of DevOps principles. To meet the demands of the modern digital market, companies must ship software extremely fast with the same quality and functionality as before. The intent here is to always have a deploy-ready, bug-free product that has passed through requisite tests.  To achieve this, developers and testers must use a CI/CD pipeline. It ensures the creation and release of software in a fast, secure, and reliable way. Like Agile Development and DevOps, CI/CD requires the right tools, frameworks, and skilled personnel to succeed and present the desired technical and commercial results. After all, humans don’t scale, but machines do.  A simple CI/CD pipeline involves these steps:   * A patch or new feature for a product is approved for and assigned a priority for implementation. * A human implements the patch or new feature and sends changes to a version-control system. * Submitted changes start automated build processes for the product. * If build processes are successful, automated tests are run against the product. * If all tests are successful, a release package is produced. * If packaging processes are successful, the release package is delivered to the production systems. * Package deliveries begin automated installation or upgrade processes for the product on the production systems. * Over time, operations personnel implement and automate monitoring capabilities for the product. * A human observes the product's functionality and requests a patch or a new feature in the product. This request is fed back to step 1. |  |
|  | Black -box Testing | [\*](#SubText1)From Wikipedia [[7]](#RefSuperText7)  Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system, and user acceptance. It is sometimes referred to as specification-based testing. |  |
|  | White -box Testing | [\*](#SubText1)From Wikipedia [*[8]*](#RefSuperText8)  White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of software testing that tests the internal structures or workings of an application at the level of its source code, as opposed to its functionality (i.e., black-box testing). The essence of white-box testing is the careful testing of the application at the source code level to reduce hidden errors. These tests exercise every visible path of the source code to minimize errors and create an error-free environment. The whole point of white-box testing is the ability to know which line of the code is being executed at any point and being able to discover what the correct output should be.  White-box testing can be applied at the unit, integration, and system levels of the software testing process. In whitebox testing, testers use an internal perspective of the system to design test cases. As programming skills are involved, the tester is a software design engineer in test (SDET). The tester chooses inputs to exercise paths through the code and figure out the expected outputs. White-box test cases derive from design techniques including control flow testing, data flow testing, branch testing, path testing, statement coverage and decision coverage as well as modified condition/decision coverage. White-box testing is the use of these techniques as guidelines to create an error-free environment by examining all code. These white-box testing techniques are the building blocks of white-box testing. |  |
|  | Toolchains | [\*](#SubText1)From [Wikipedia](https://en.wikipedia.org/wiki/DevOps_toolchain) [[9]](#RefSuperText9)  A DevOps or DevSecOps toolchain is a set or combination of tools that aid in the delivery, development, and management of software applications throughout the systems development life cycle, as coordinated by an organization that uses DevOps practices. DevOps tools fit into one or more activities, which support specific DevOps initiatives: Plan, Create, Verify, Package, Release, Configure, Monitor, and Version Control.  In general, the tools forming a toolchain run one after the other so the output or resulting environment state of each tool becomes the input or starting environment for the next one. As DevOps is a set of practices that emphasizes the collaboration and communication of both software developers and other information technology (IT) professionals while automating the process of software delivery and infrastructure changes, its implementation can include the definition of the series of tools used at various stages of the lifecycle. But because DevOps is also a cultural shift and collaboration between development and operations, there is no such thing as a single DevOps tool. Instead, developers use a collection of tools, potentially from a variety of vendors, in one or more stages of the lifecycle. Similarly, there is no such thing as a single or standard DevOps toolchain. Every business adopting DevSecOps practices sets up its DevSecOps toolchain depending on its own unique needs. |  |
|  | Branching | [\*](#SubText1) |  |
|  | Open-Source | [\*](#SubText1) |  |
|  | Rest Services | [\*](#SubText1) |  |
|  | CRUD | [\*](#SubText1) |  |
|  | Single Source of Truth | [\*](#SubText1) |  |
|  | Bug- and Task Tracking System | [\*](#SubText1) |  |
|  | SSO | [\*](#SubText1) From [Wikipedia](https://en.wikipedia.org/wiki/Single_sign-on) [[16]](#RefSuperText16)  Single sign-on (SSO) is an authentication scheme that allows a user to log in with a single ID and password to any of related, yet independent, software systems. Single sign-on refers to systems where a single authentication provides access to multiple applications by passing the authentication token seamlessly to configured applications. True single sign-on allows the user to log in once and access services without re-entering authentication factors. |  |
|  | SAML | [\*](#SubText1) From [Wikipedia](https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language) [[17]](#RefSuperText17)  Security Assertion Markup Language (SAML) is an open standard for exchanging authentication and authorization data between parties, like between an identity provider and a service provider. SAML is an XML-based markup language for security assertions (statements that service providers use to make access-control decisions). SAML is also:   * A set of XML-based protocol messages * A set of protocol message bindings * *A set of profiles (using all of the above)*   *Benefits of using single sign-on include:*   * *Mitigate risk for access to third-party sites ("federated authentication") because user passwords are not stored or managed externally* * *Reduce password fatigue from different username and password combinations* * *Reduce time spent re-entering passwords for the same identity* * *Reduce IT costs due to lower number of IT help desk calls about passwords*   *SSO shares centralized authentication servers that all other applications and systems use for authentication purposes and combines this with techniques to ensure that users do not have to actively enter their credentials more than once.* |  |
|  | User Community | [\*](#SubText1) |  |

Notes

* A quote edited for clarity

References

1. *Version Control – Wikipedia*, [Version Control on Wikipedia](https://en.wikipedia.org/wiki/Version_control)
2. *Repository (version control) – Wikipedia*, [Source Code Repository](https://en.wikipedia.org/wiki/Source_code_repository)
3. *DevOps – Wikipedia,* [*DevOps*](https://en.wikipedia.org/wiki/DevOps)

*DevOps – New Relic,* [*What is DevOps?*](https://newrelic.com/devops/what-is-devops)

*DevOps – Gartner,* [*Gartner glossary DevOps*](https://www.gartner.com/en/information-technology/glossary/devops)

*DevOps – Cyclr,* [*DevOps and Automation: What is it, and how can implementing it help?*](https://cyclr.com/blog/devops-and-automation)

1. *DevSecOps – Wikipedia,* [*DevSecOps, Shifting Security Left*](https://en.wikipedia.org/wiki/Shift-left_testing)

*DevSecOps – Red Hat,* [*What is DevSecOps?*](https://www.redhat.com/en/topics/devops/what-is-devsecops)

*DevSecOps – Wikipedia, DevOps - Wikipedia*

1. *Remote-First* – Adapted from article on twist.com: [“Remote-ish: The Remote-First Solution for Hybrid Companies”](https://twist.com/remote-work-guides/remote-hybrid-companies)
2. *CI/CD* – Adapted from blog by CloudScaleQA, [Continuous Integration & Deployment (CI-CD): The Future Paradigm has arrived](https://medium.com/cloudscaleqa/continuous-integration-deployment-ci-cd-the-future-paradigm-has-arrived-8776ba34439e)

*CI/CD* – Adapted from blog post by [Joseph Matthias Goh](https://joeir.medium.com/), [That CI/CD Thingy: Principles, Implementation & Tools](https://blog.gds-gov.tech/that-ci-cd-thing-principles-implementation-tools-aa8e77f9a350)

*CI/CD* – Adapted from article, SEI [SEI > Publications > Blog > A Framework for DevSecOps Evolution and Achieving Continuous-Integration/Continuous-Delivery (CI/CD) Capabilities](https://insights.sei.cmu.edu/blog/a-framework-for-devsecops-evolution-and-achieving-continuous-integrationcontinuous-delivery-cicd-capabilities/) by [Lyndsi Hughes](https://insights.sei.cmu.edu/authors/lyndsi-hughes/) and [Vanessa Jackson](https://insights.sei.cmu.edu/authors/vanessa-jackson/)

1. *Black-box Testing –* Wikipedia, [Black box Testing on Wikipedia](https://en.wikipedia.org/wiki/Black-box_testing)
2. *White-box Testing –* Wikipedia, [White box testing on Wikipedia](https://en.wikipedia.org/wiki/White-box_testing)
3. *Toolchain* – Wikipedia, [DevOps toolchain on Wikipedia](https://en.wikipedia.org/wiki/DevOps_toolchain)
4. *.*
5. *.*
6. *.*
7. *.*
8. *.*
9. .
10. *SSO* – Wikipedia, [Single Sign-On, on Wikipedia](https://en.wikipedia.org/wiki/Single_sign-on)
11. *SAML* – Wikipedia, [Security Assertion Markup Language on Wikipedia](https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language)
12. *.*
13. *.*
14. *.*