Using automation tools for code quality improvements for Java applications

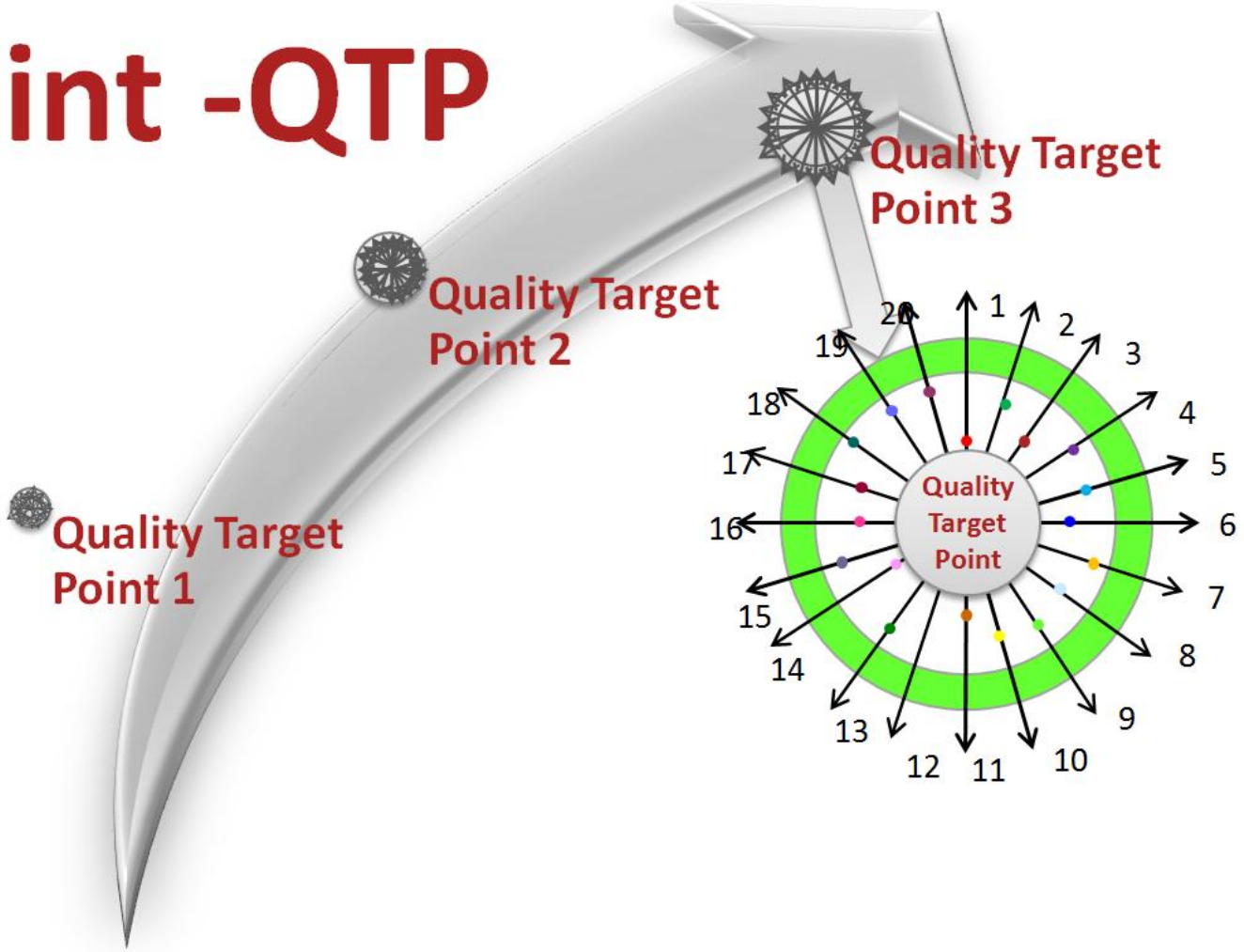
Alexander Lipanov
CEO of Softarex Technologies Inc,
PhD in Applied Mathematics
alex@softarex.com

Alexander Chelombitko
Head of Department,
Softarex Technologies Inc
achelombitko@softarex.com

Hands-on Lab Plan

- **Quality Target Point. Base concepts**
  - Definition of Quality Target Point
  - Best software product definition through set of Quality Target points
  - Parameters of Quality Target Point and its measures: Statistical metrics; Object oriented metrics; CISQ measures for quality characteristics; Additional parameters
  - Organization of development process with Quality Target Point
- **codeNforcer introduction**
  - Features
  - Typical Workflow of codeNfrocer
  - Some examples of violations in code
  - Getting Started with codeNforcer. Practical work with codeNforcer
- **Conclusion**

Quality Target Point -QTP



SEND YOUR QUESTIONS TO JAVAONE@SOFTAREX.COM

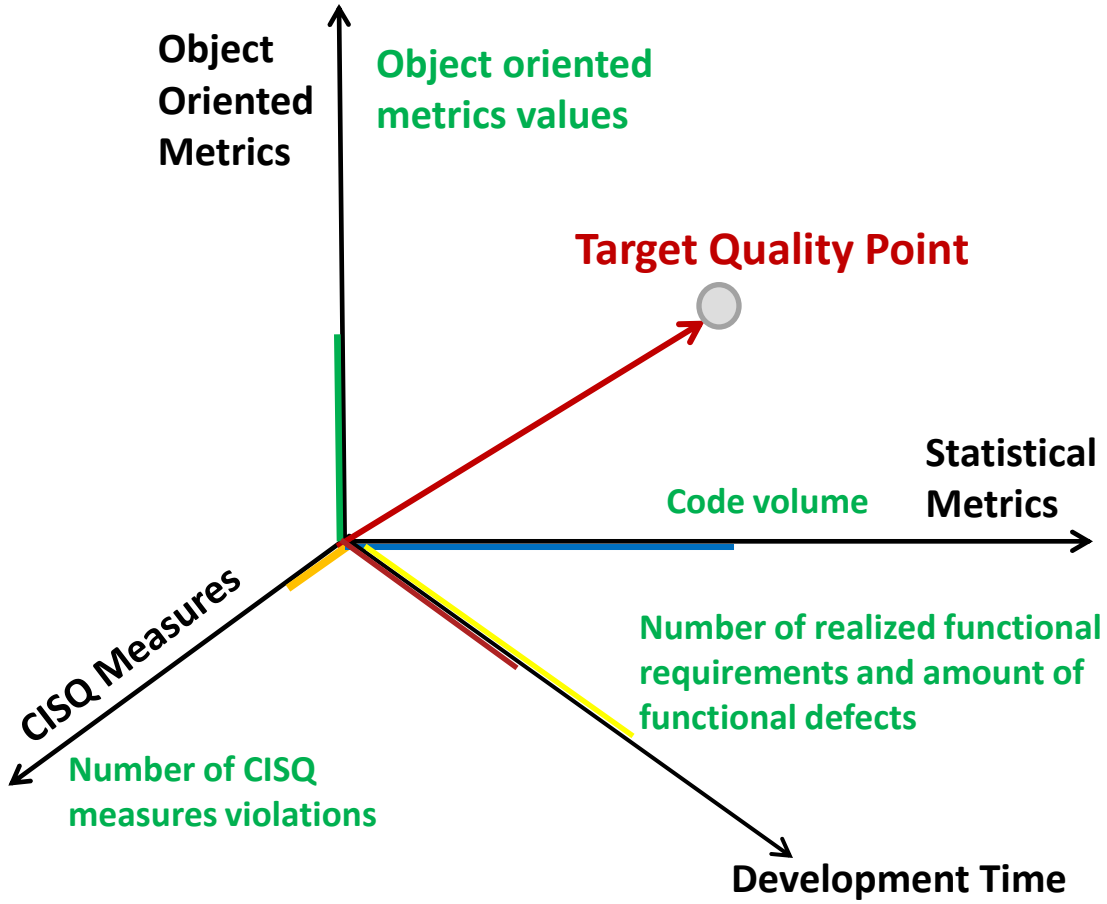
Base Definition of Quality Target Point (QTP)

Key factors influencing to Software Quality:

- Correspondence to Functional Requirements
- Code Architecture and organization
- Non functional requirements - Reliability, Performance Efficiency, Maintainability, Security
- Size of source code

Quality Target Point – abstract point where realized necessary **Functional Requirements** and achieved following conditions:

1. Values of **Object Oriented Metrics** ∈ [*intervals of recommended values*]
2. Number **of CISQ Measures violations** → **0**
3. **Source code volume have minimal size** and provide all necessary functionality. At least source code not have duplications
4. Number of defects in functionality meet to planned values which allow use system by users
5. Quality Target Point has exact defined date



Base Parameters of QTP and its Measures

Development time

Functional Requirements

↓

Number of Requirements

↓

Number of defects in realization of Functional requirements

Software Source Code Architecture

↓

Object Oriented Metrics

Level of internal connections of types	• Lack of Cohesion, Lack of Cohesion (Henderson-Sellers, Chidamber & Kemerer, etc)
Level of external connections of types	• Efferent Coupling, Instability, Abstractness, Distance from the Main Sequence
Level of namespaces and packages	• Coupling, Association Between Classes, Afferent Coupling, Relational Cohesion

Source code volume

↓

Statistical metrics (number of lines, classes, methods, packages, namespaces, etc)

Non functional requirements

↓

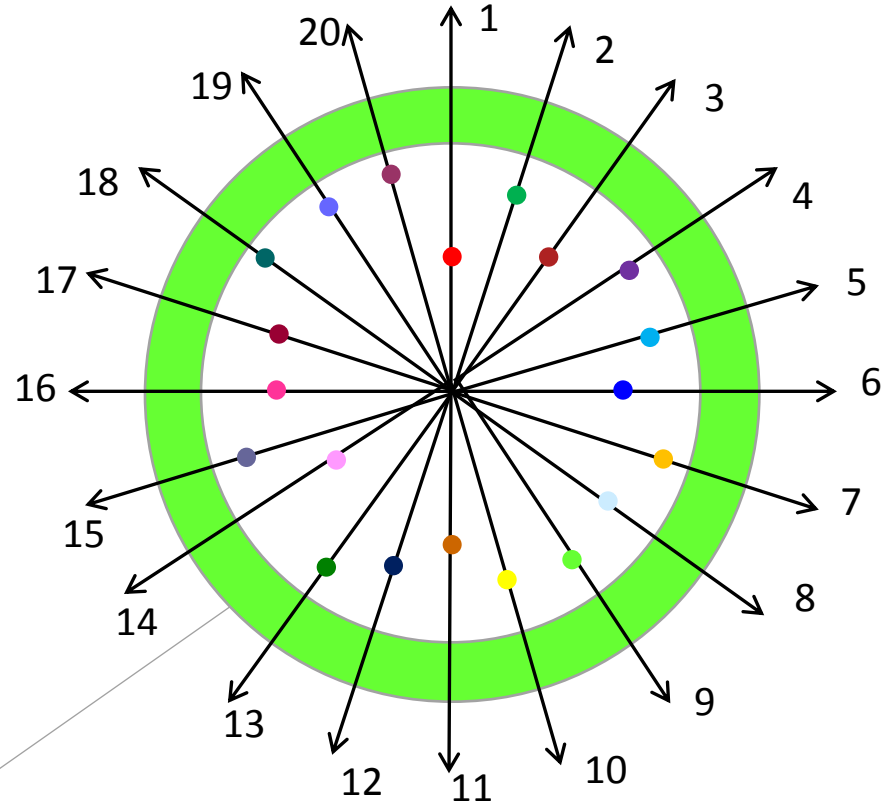
CISQ Code quality measures

- **Reliability**
- **Performance Efficiency**
- **Maintainability**
- **Security**

N-Dimensional Representation of Quality Target Point

Quality Target Point in N-Dimensional Representation.

Dots on chart is current values of parameters



Green area – area where values of QTP parameters are acceptable or just best if its value equal Radius of Green Circle

Possible list of parameters for Quality Target Point

Object Oriented Metrics	**Level of internal connections of types** 1. Lack of Cohesion 2. Lack of Cohesion (Henderson-Sellers, Chidamber & Kemerer, etc) **Level of external connections of types** 3. Efferent Coupling 4. Instability 5. Abstractness 6. Distance from the Main Sequence **Level of namespaces and packages** 7. Coupling 8. Association Between Classes 9. Afferent Coupling 10. Relational Cohesion
CISQ Measures	11. Reliability 12. Performance Efficiency 13. Maintainability 14. Security
Statistical Metrics	15. Number of lines of code 16. Number of methods 17. Number of realized functional requirements 18. Amount of functional defects 19. Mean time between failures 20. …….

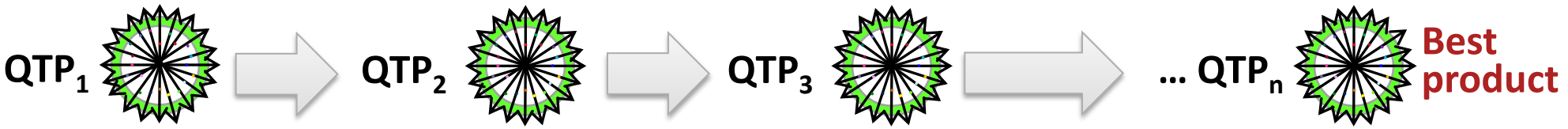
Best Software Product is a Quality Target Point with best values

Best software product is a software product which have final (last) **Quality Target Points** where all parameters equal to their planned values or have acceptable deviations

$$\textbf{Best product} = QTP_n(p_i=v_i,\ p_{i+1}=v_{i+1},\ldots,\ p_j=v_j)$$

Where $\mathbf{p_i}$ – parameter in QTP, $\mathbf{v_i}$-planned value of parameter $\mathbf{p_i}$ on QTP_n, $i \in [1,j]$, j – number of parameters in QTP

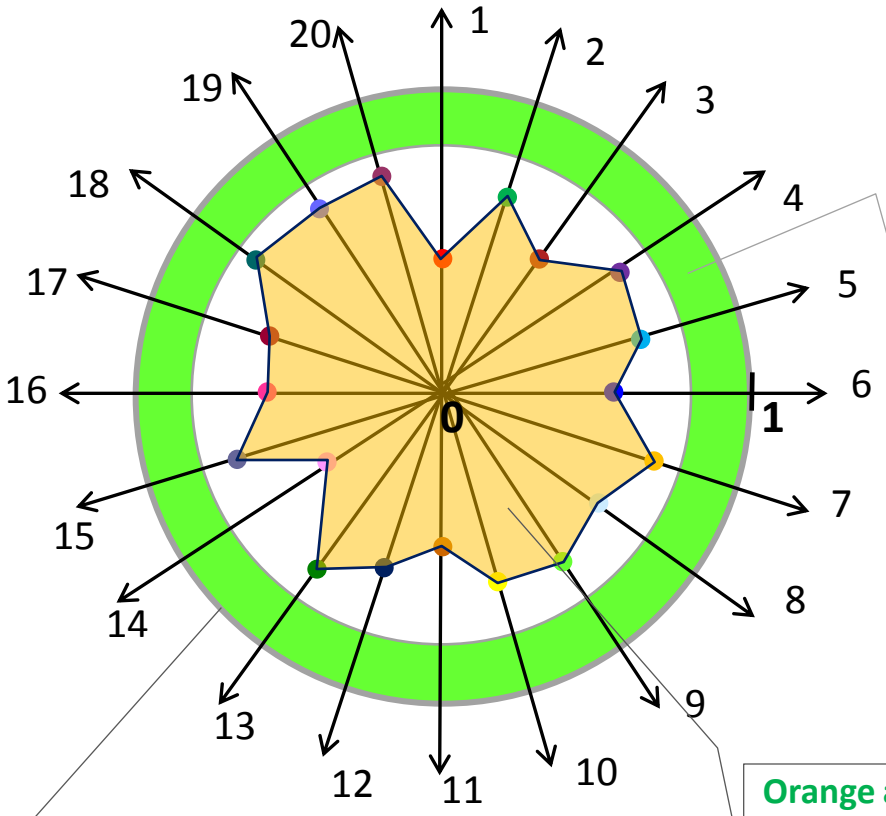
Number of parameters not achieved on final QTP and its deviations from planned values show general quality of your product and how far it from ideal/best/good/acceptable state



Product Quality Coefficient based on QTP

Quality Target Point in N-Dimensional Representation.

Dots on chart is current values of parameters



Green area – area where values of QTP parameters are acceptable or just best if its value equal to Radius of Green Circle (In our case Radius = 1 basing on our measure system). **S- Square of QTP circle when all parameters have best values. S=3.14 in case if Radius =1**

Grey Circle–QTP circle, maximal value of parameters

Orange area – area covered by current values of QTP parameters. **Square of this area is S_{QTP}**

Important conclusions

1. Lets define ***k*** as Coefficient of Product Quality:

$$k = 1 - \frac{S_{QTP}}{S}$$

2. Value of ***k*** can be calculated basing on square of area defined by values of QTP parameters
3. If ***k=0*** then we have situation:

Best product = $QTP_n(p_i=1, p_{i+1}=1,..., p_j=1)$

4. Than smallest value have ***k*** than better product quality we have and vise versa
5. With this approach basing on time spent for transition from QTP_n to QTP_{n+1} it possible estimate amount of man-hours and cost for this transition. Information about time can be taken from project management system.

codeNforcer is a tool for measuring parameters for QTP

Web based code analysis and code improvements system. System accessible in public or corporate cloud

Source code analysis and improvements

- Supported programming languages: Java. Support for C++, C#, PHP and Objective C coming soon
- Source code statistics collection
- Source code checking basing on schedule
- Code validation for Reliability, Efficiency, Maintainability, Security improvements basing on CISQ measures and other rules
- Object Oriented Metrics calculation
- Recommendations for source code improvements basing on analysis of Object Oriented Metrics
- Code convention checking
- Code checking basing on user's rules
- Recommendations for source code improvements basing on CISQ measures for Reliability, Efficiency, Maintainability, Security

Team work and integrations

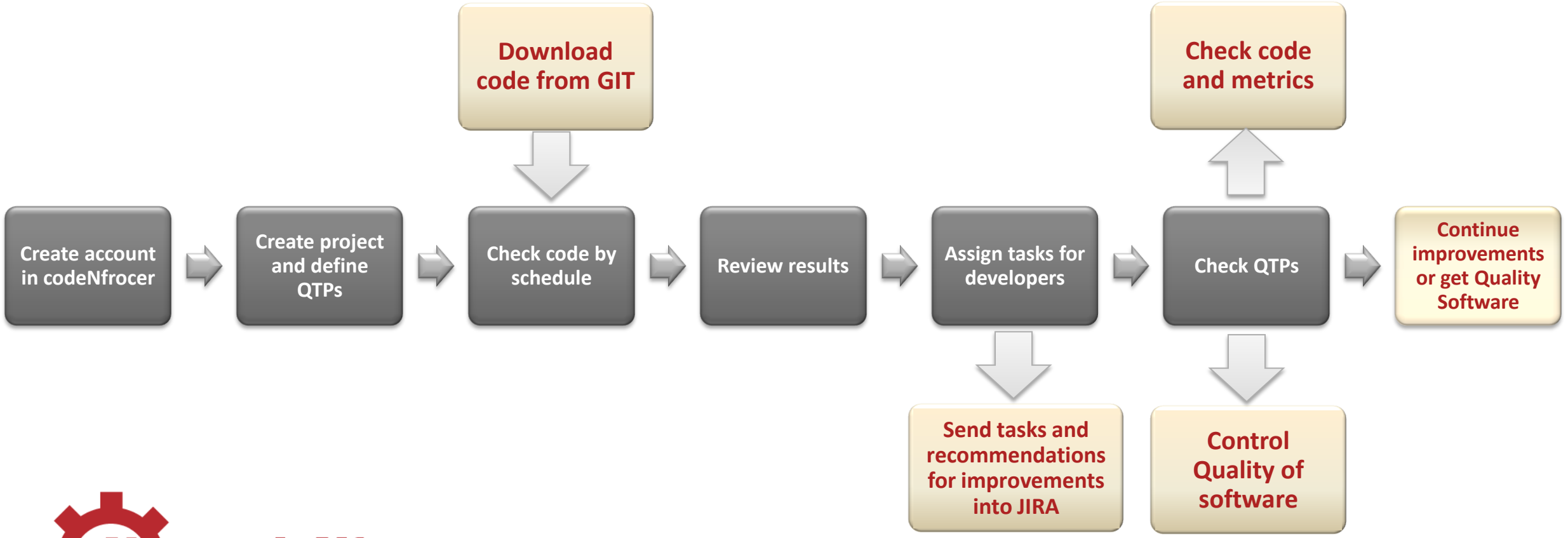
- Creating project's groups
- Web based tools for source code review
- Users and Projects management
- Integration with SVN, GIT and TFS
- Integration with JIRA for interaction on level of users, projects and SCRUM dashboards
- Team notifications by email
- Loading projects, users and statistics necessary for QTP from JIRA
- Assigning tasks for developers in JIRA for source code improvements basing on generated recommendations

Measurements and Reports

- Source code statistic including weekly and monthly analysis
- Creation, management and tracking of QTPs for projects
- Metrics calculations and their changes dynamics (reflected in QTP)
- Tracking of improvements progress for Performance, Reliability, Maintainability, Security violations

SEND YOUR QUESTIONS TO JAVAONE@SOFTAREX.COM

Typical Workflow of codeNfrocer



codeNforcer.com

SEND YOUR QUESTIONS TO JAVAONE@SOFTAREX.COM

Example of code with violation for Security

ASCSM-CWE-397 : Declaration of Throws for Generic Exception

Description: Software unaware of accurate execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors

Example of Code:

```
public class DaoManager {
  ...
public static Session getSession() throws Exception {
    if (FacesContext.getCurrentInstance() == null) {
      throw new Exception("DaoManager: 44;
\"FacesContext.getCurrentInstance() is null\"");
    }
    return PersistenceSessionManager.getBean().getSession();
  }
  ...
}
```

Example of code with violation for Security

ASCSM-CWE-396 : Declaration of Catch for Generic Exception

Description: Software unaware of accurate execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors

Example of Code:

```
public class FieldListBean extends BasePageBean implements
Serializable {
  ...
 protected void onConstruct() {
    try {
      fields =
DaoManager.query().from(QField.field).fetchAll().list(QField.field);
    } catch (Exception e) {
      log.error(e.getMessage(), e);
    }
  }
  ...
}
```

Example of code with violation for Reliability

ASCCRM-RLB-14: Parent Class Element with References to Child Class Element

Description: Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors

Example of Code:

```
public class TestBean extends BaseValidationBean implements Serializable
{
...
}

public abstract class BaseValidationBean extends BasePageBean {
...
}

public abstract class BasePageBean implements Serializable {
...
  public void submit() {
    try {
      ((TestBean) this).onSubmit();
      HeaderBean.updateResponsesCount();
    } catch (Exception e) {
    }
  }
}
```

Example of code with violation for Reliability

ASCCRM-RLB-8: Named Callable and Method Control Elements with Variadic Parameter Element

Description: Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Example of Code:

```
public class RedirectHelper extends BaseBaseClass {
  ...
  public static void sendRedirectWithParam(String linkTemplate, Object...
      params) {
    String link = String.format(linkTemplate, params);
    sendRedirect(link);
  }
  ...
}

public class FieldEditBean extends BaseValidationBean implements Serializable
      {
  ...
  private void afterSave() {

      RedirectHelper.sendRedirectWithParam(RedirectHelper.getLink(PagesType
      s.FIELD_LIST));
  }
  ...
}
```

Example of code with violation for Performance

ASCCPEM-PRF-2: Immutable Storable and Member Data Element Creation

Description: Software featuring known under efficient coding practices requires excessive computational resources

Example of Code:

```
public class TestBean extends BaseValidationBean implements Serializable {
  ...
}

public abstract class BaseValidationBean extends BasePageBean {
  ...
}

public abstract class BasePageBean implements Serializable {
  ...

  public void submit() {
    try {
      ((TestBean) this).onSubmit();
      HeaderBean.updateResponsesCount();
    } catch (Exception e) {
    }
```

Getting Started with codeNforcer

1. Open www.codenforcer.com
2. Create your Trial account
3. Create Project and use code for it from **https://github.com/SoftarexTechnologies/javaOne_analysis.git**
4. Define values for Quality Target Points
5. Analyze code and review of problems and recommendations for its fixing
6. **Lets fix some problems and will see how changed QTP**

codeNforcer.com

SEND YOUR QUESTIONS TO JAVAONE@SOFTAREX.COM

Conclusion

- We have reviewed approach for software quality control and improvements based on QTP
- We have studied what metrics and measures can be used for code improvements and what its means
- It is reviewed few examples of code with CISQ measures violations
- Reviewed how to use codeNfrocer and tried it for code analysis and conducted improvements in code basing on recommendations from codeNfrocer
- **Try and use online www.codenforcer.com**

codeNforcer.com

SEND YOUR QUESTIONS TO JAVAONE@SOFTAREX.COM

Thank You for Your Attention

Softarex Technologies, Inc.

Headquarters

901 N. Pitt Street, Suite 320
Alexandria, VA 22314, USA
Tel: +1 (703) 836 18 60
E-mail: info@softarex.com