

Тестирование “белым ящиком”

Этот метод изучает не только внешнее поведение программы, но и ее внутреннее устройство (исходные тексты).

Проектирование тестов основано на изучении логики программы.

Тесты проектируются таким образом, чтобы каждая команда условного перехода выполнялась в каждом направлении хотя бы один раз. Проверить каждый путь, каждую ветвь алгоритма.

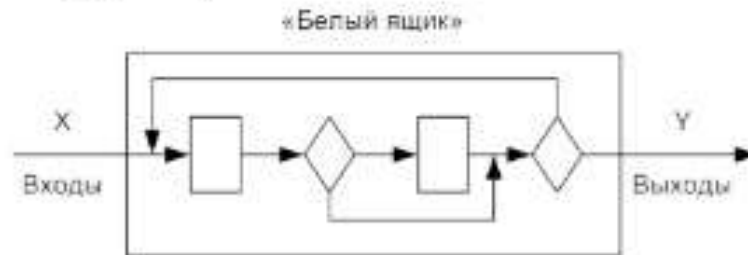
Задание: СОСТАВИТЬ ОПТИМАЛЬНЫЙ набор

- ▶ «ВИСОКОСНЫЙ ГОД» - КОГДА ГОД делится на 100 без остатка, но не делится на 400 без остатка

```
public class LeapYearChecker {  
    public static void main(String[] args) {  
        Scanner scanner = new  
Scanner(System.in);  
  
        System.out.print("Введите год: ");  
        int year = scanner.nextInt();  
  
        boolean isLeapYear = year % 4 == 0  
&& (year % 100 != 0 || year % 400 == 0);  
  
        if (isLeapYear) {  
            System.out.println(year + "  
является високосным годом.");  
        } else {  
            System.out.println(year + " не  
является високосным годом.");  
        }  
    }  
}
```

Тестирование белого ящика

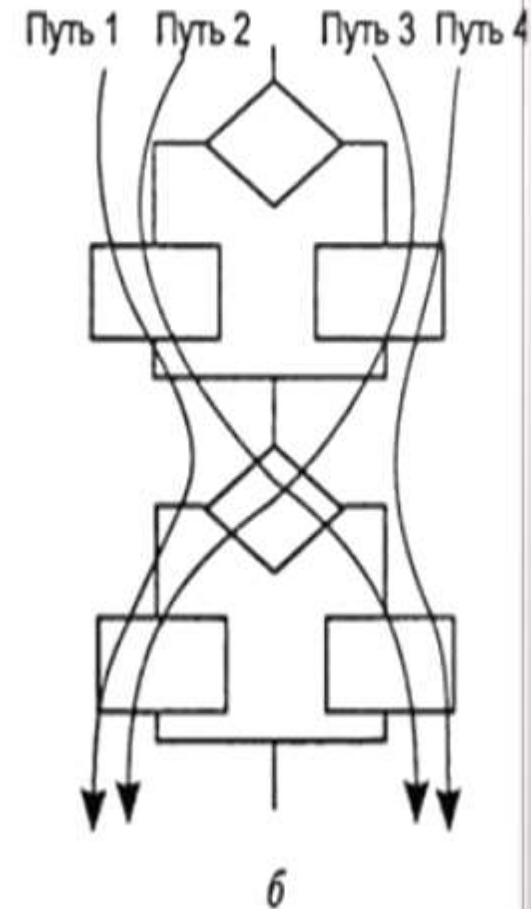
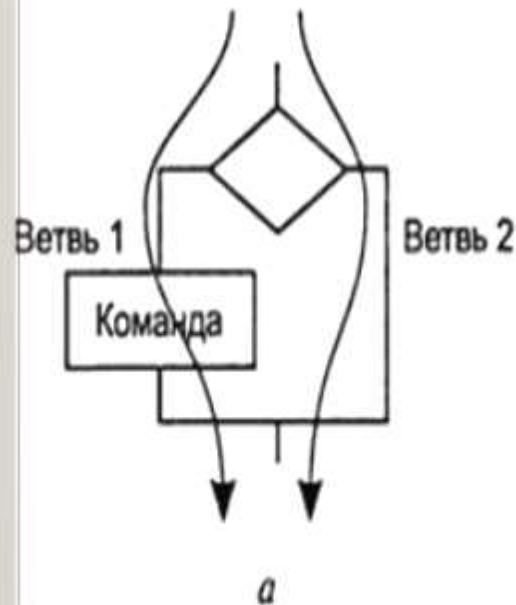
- вид тестирования, в котором тестировщик должен обладать знаниями о внутренних механизмах работы продукта и иметь доступ к исходному коду приложения



- покрытие тестами каждой строчки кода системы

методы тестирования по принципу «белого ящика»

- покрытие операторов;
- покрытие решений (путей);
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.



Метод покрытия операторов

- ▶ выполнение каждого оператора программы хотя бы один раз

```
float div_by_zero_test(int  
x) {
```

```
1. int tmp = 0;
```

```
2. if (x > 0) tmp = 1;
```

```
3. float b = 1.0 / tmp;
```

```
return b;
```

Исходно е значени е	Ожидае мый результ ат	Фактиче ский результ ат	Результ т теста	
100	1.f			

Метод переходов (решений)

- ▶ чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз.

```
float div_by_zero_test(int x) {
```

```
1. int tmp = 0;
```

```
2. if (x > 0) tmp = 1;
```

```
3. float b = 1.0 / tmp;
```

```
return b; }
```

Исходное значение	Ожидаемый результат	Фактический результат	Результат теста	
-3	ошибка			



ерия
личество и
результат
словия принимал
и «ложь», по
раз

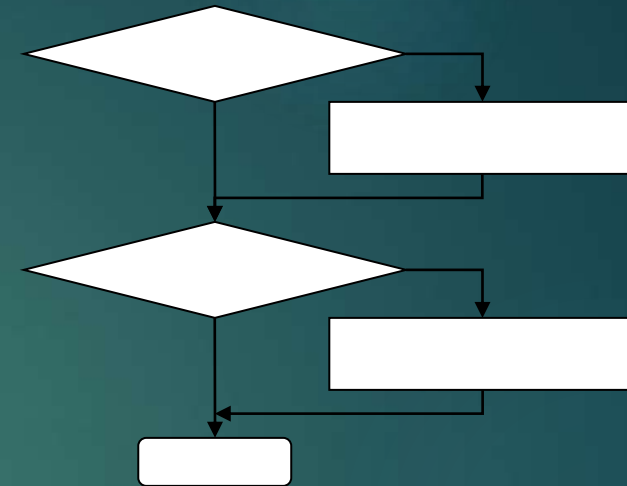
Пример

8

```
1 public void Method (ref int x) {  
2   if (x>17)  
3     x = 17-x;  
4   if (x== -13)  
5     x = 0;  
6 } (vx, вых) = {(30, 0)} - все операторы трассы 1-2-3-4-5-6
```

критерий ветвей (**C1**):
(vx, вых) = {(30, 0),
(17, 17)}

критерий путей (**C2**):
(vx, вых) = {(30,0), (17,17),
(-13,0), (21,-4)}



Условия операторов if

	(30,0)	(17,17)	(-13,0)	(21,-4)
2 if (x>17)	>	<=	<=	>
4 if (x== -13)	=	!=	=	!=

Покрытие условий

9

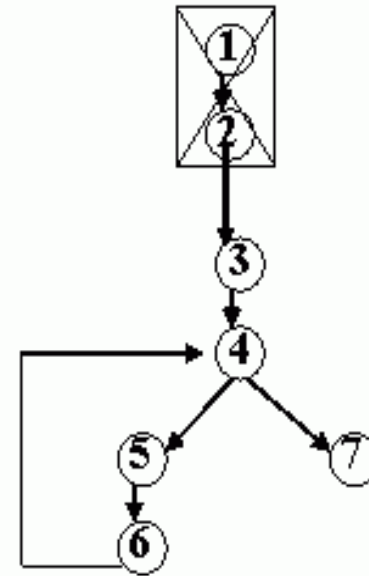
- ▶ Формирование числа тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней мере, один раз.

```
if(year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)) a++2;
```

Пример: тестирование путей

/* Функция вычисляет неотрицательную
степень n числа x */

```
1 double Power(double x, int n){  
2   double z=1; int i;  
3   for ( i = 1;  
4     i <= n;  
5     i++ )  
6     { z = z*x; } /* Возврат в п.4 */  
7   return z;  
}
```



Управляющий граф
программы

примеры путей: (3,4,7), (3,4,5,6,4,5,6), (3,4), (3,4,5,6)

примеры ветвей: (3,4) (4,5,6,4) (4,7)

Формирование тестовых наборов для тестирования маршрутов может осуществляться по нескольким критериям:

- покрытие операторов;
- покрытие решений (переходов);
- покрытие условий;
- покрытие путей;
- **комбинаторное покрытие условий.**

Комбинаторное покрытие условий

12

Этот критерий требует создания такого множества тестов, чтобы все возможные комбинации результатов условий в каждом решении и все операторы выполнялись, по крайней мере, один раз.

Тестирование по потоку данных

Данные: переменная x – Проверка

▶ объявление (declaration): **$int\ x;$**

▶ определение (definition, d-use): **$x = 99;$**

▶ использование в вычислениях (computation use, c-use): **$z = x + 1;$**

▶ использование в условиях (predicate use, p-use):

$if\ (x > 17)\ \{ \dots \};$

▶ удаление (kill, k-use):

$x = null;$

▶ на основе всех объявлений - проверяется, что для каждой переменной существует путь от каждого её определения к её использованию в вычислениях.

▶ ветвлений на основе всех объявлений — проверяется, что для каждой переменной существует путь от каждого её определения к её использованию в условиях.

▶ всех вычислений и ветвлений — что для каждой переменной существует хотя бы один путь от каждого её определения к каждому её использованию в вычислениях и в условиях.

▶ всех объявлений и всех путей без пере-объявлений (без циклов или с однократными повторениями циклов) — для каждой переменной проверяются все пути от каждого её определения к каждому её использованию в вычислениях и в условиях

**Входные данные
определяются...**

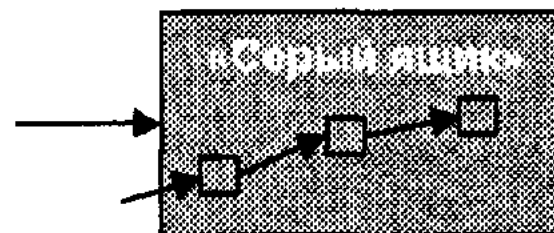
Результат

Требованиями



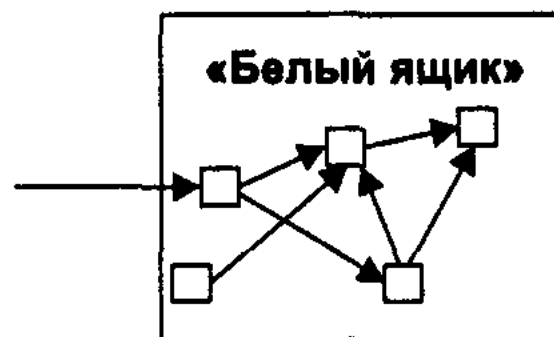
*Полученные выходные
данные в сравнении
с требуемыми
выходными данными*

*Требованиями
и ключевыми
элементами
проектирования*



*Как для тестирования
«черного» и «белого ящика»*

*Элементами
проектирования*



*Подтверждение
ожидаемого поведения*

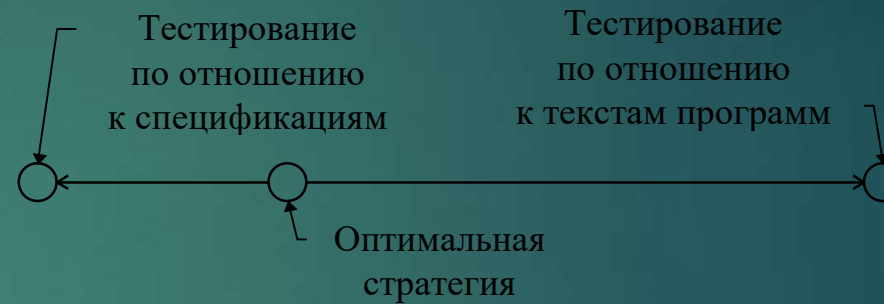
Сравнение методов тестирования

16

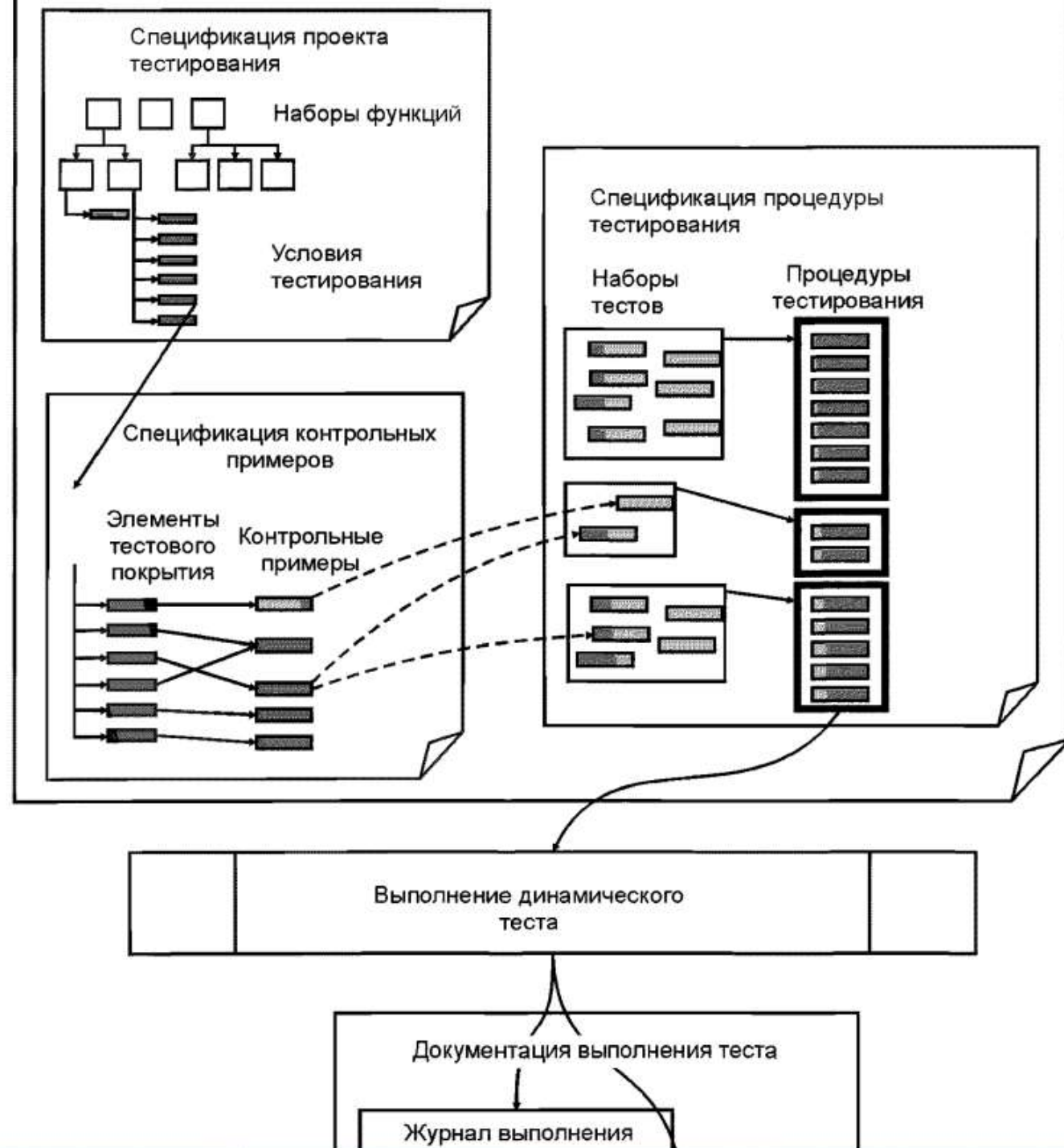
Критерий	Black Box	White Box
<i>Определение</i>	тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы	тестирование, основанное на анализе внутренней структуры компонента или системы
<i>Уровни, к которым применима техника</i>	В основном: <ul style="list-style-type: none">• Приемочное тестирование• Системное тестирование	В основном: <ul style="list-style-type: none">• Юнит-тестирование• Интеграционное тестирование
<i>Кто выполняет</i>	Как правило, тестировщики	Как правило, разработчики
<i>Знание программирования</i>	Не нужно	Необходимо
<i>Знание реализации</i>	Не нужно	Необходимо
<i>Основа для тест-кейсов</i>	Спецификация, требования	Проектная документация

Оценка тестирования

- ❑ Оценить покрытие функциональности тестами
- ❑ Оценить покрытие кода
- ❑ Проанализировать дефекты
- ❑ Определить, были ли достигнуты критерии завершенности и успешности тестирования



Тестовое Покрытие - метрика оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.



Тест-кейс

- ▶ набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Под тест-кейсом также может пониматься соответствующий документ, представляющий формальную запись тест-кейса

Общий вид тест-кейса

Идентификатор	Приоритет	Связанное с тест-кейсом требование	Заглавие (суть) тест-кейса		Ожидаемый результат по каждому шагу тест-кейса
UG_U1.12	A	R97	Галерея	Загрузка файла	Галерея, загрузка файла, имя со спец-символами Приготовление: создать непустой файл с именем #\$\$%^&.jpg. 1. Нажать кнопку «Загрузить картинку». 2. Нажать кнопку «Выбрать». 3. Выбрать из списка подготовленный файл. 4. Нажать кнопку «ОК». 5. Нажать кнопку «Добавить в галерею».
Модуль и подмодуль приложения					
Исходные данные, необходимые для выполнения тест-кейса					
Шаги тест-кейса					
			1. Появляется окно загрузки картинки. 2. Появляется диалоговое окно браузера выбора файла для загрузки. 3. Имя выбранного файла появляется в поле «Файл». 4. Диалоговое окно файла закрывается, в поле «Файл» появляется полное имя файла. 5. Выбранный файл появляется в списке файлов галереи.		

! Нет и не может быть ожидаемого результата в виде «приложение вызывает ошибку в операционной системе и аварийно завершается с потерей всех пользовательских данных».

Четкие формулировки шагов

- ▶ «Установить приложение на диск C».
- ▶ «Нажать на иконку приложения».
- ▶ «Окно приложения запустится».
- ▶ «Работает верно».
- ▶ «ОК».
- ▶ «Количество найденных файлов совпадает». (С чем?)
- ▶ «Приложение отказывается выполнять команду».

Инструментальные средства: создание

QAComplete²⁹²

1 Id: (Auto Generated) 2 3 4 Folder Name: 5 6 7

Title: 8 9 10 11 12 13 14 15 16

Priority: 17 18 19

Avg Run Time: (Auto Calculated) 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Last Run Test Set: (Auto Calculated) 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200

Last Run Release: 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300

Description: 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400

Owner: 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500

Execution Type: 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600

Latest Notes: 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700

Version: 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800

Test Type: 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900

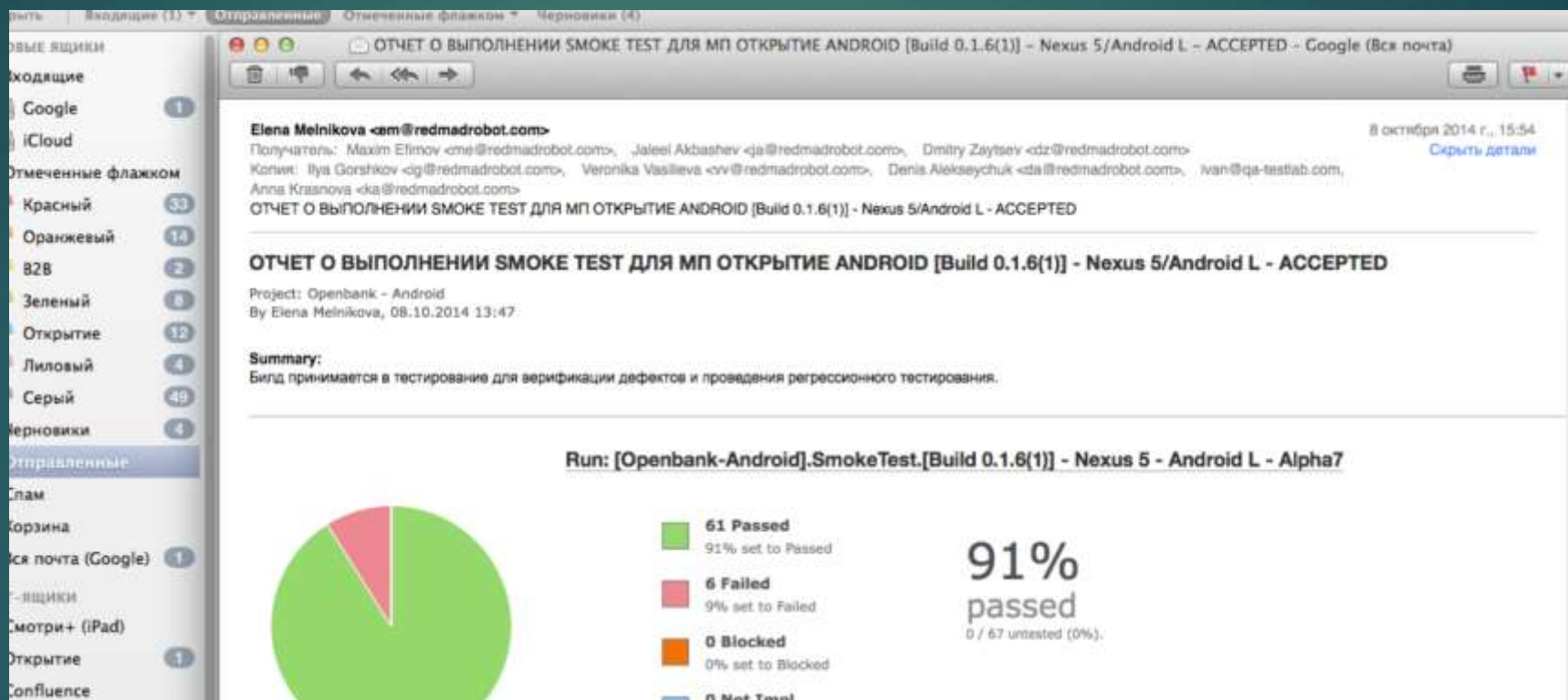
Default Host Name: 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

Linked Items: 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100

File Attachments: 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200

Reset Browse... 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300

Инструментальные средства: отчеты



Оценка выполненных тестов

Метрики покрытия/глубины тестирования

- Метрику покрытия требований (требование считается «покрытым», если на него ссылается хотя бы один тест-кейс):

$$R^{SimpleCoverage} = \frac{R^{Covered}}{R^{Total}} \cdot 100\%, \text{ где}$$

$R^{SimpleCoverage}$ — метрика покрытия требований,

$R^{Covered}$ — количество требований, покрытых хотя бы одним тест-кейсом,

R^{Total} — общее количество требований.

- Метрику плотности покрытия требований (учитывается, сколько тест-кейсов ссылается на несколько требований):

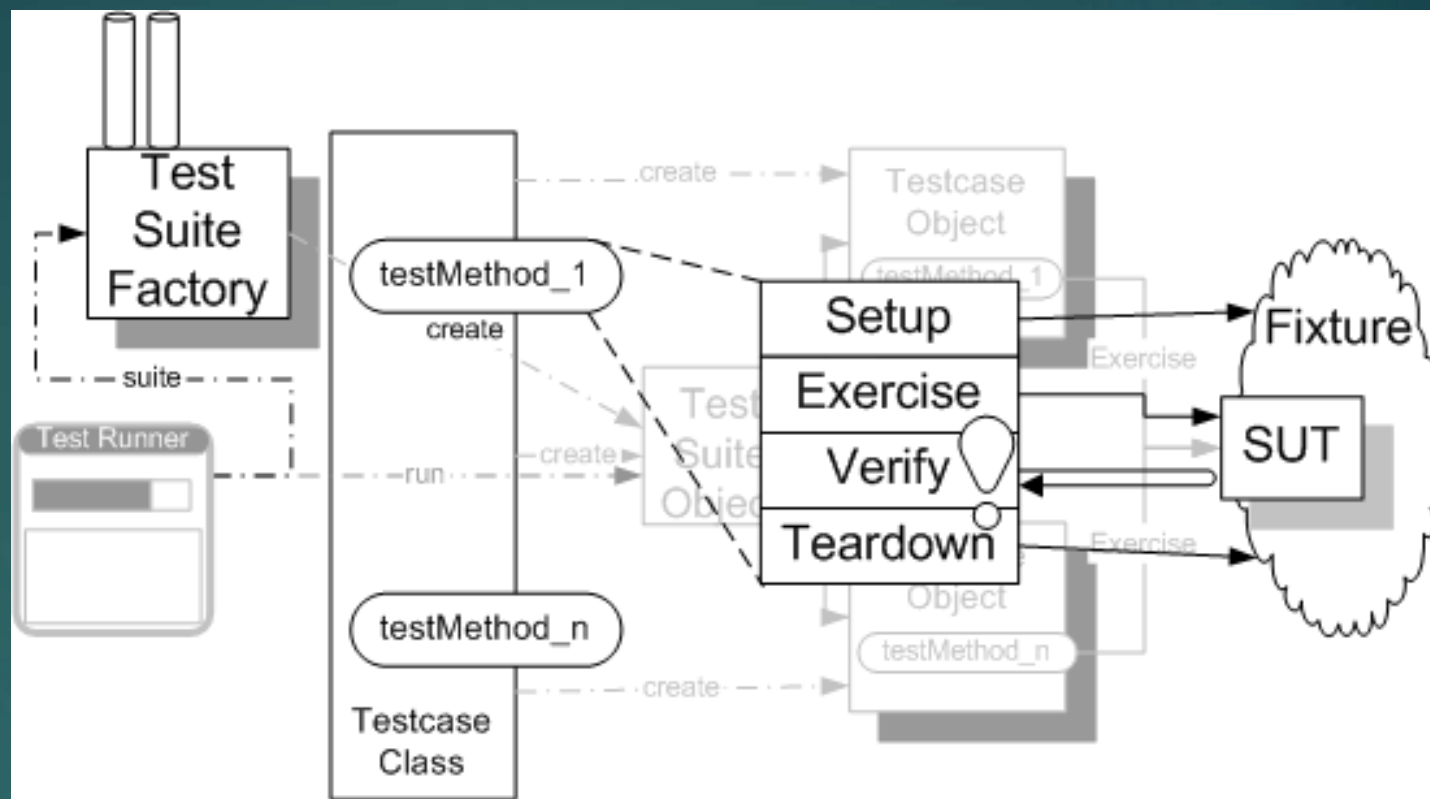
$$R^{DensityCoverage} = \frac{\sum T_i}{T^{Total} \cdot R^{Total}} \cdot 100\%, \text{ где}$$

$R^{DensityCoverage}$ — плотность покрытия требований,

T_i — количество тест-кейсов, покрывающих i -е требование,

T^{Total} — общее количество тест-кейсов,

R^{Total} — общее количество требований.



АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ

Модульное тестирование

```
public class Bmi {

    private float height, weight;

    private ICalcBmi calc;

    public Bmi(float height, float weight) {
        this.height = height;
        this.weight = weight;
        calc = new CalcBmi(); }

    public float getBMI() {
        return calc.calculate(this.height, this.weight); }

    public void setCalc(ICalcBmi calc) {
        this.calc = calc; }

    public boolean diagnosis(float b1, float b2){
        float b=getBMI();
        if(b > b1 && b< b2) return true; // значение ИМТ в интервале от b1 до b2
        else return false; }
}

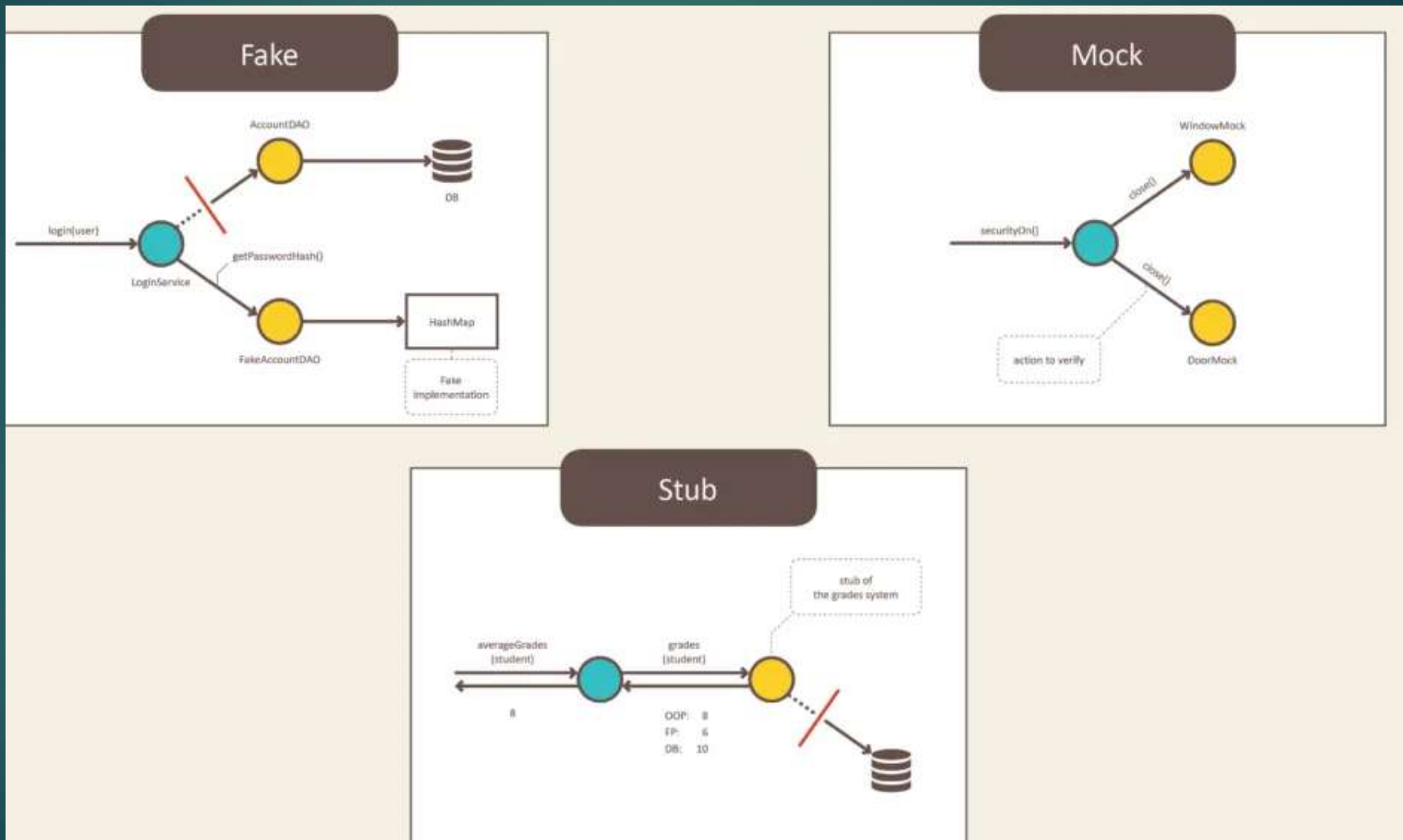
interface ICalcBmi {
    float calculate (float h, float w);}

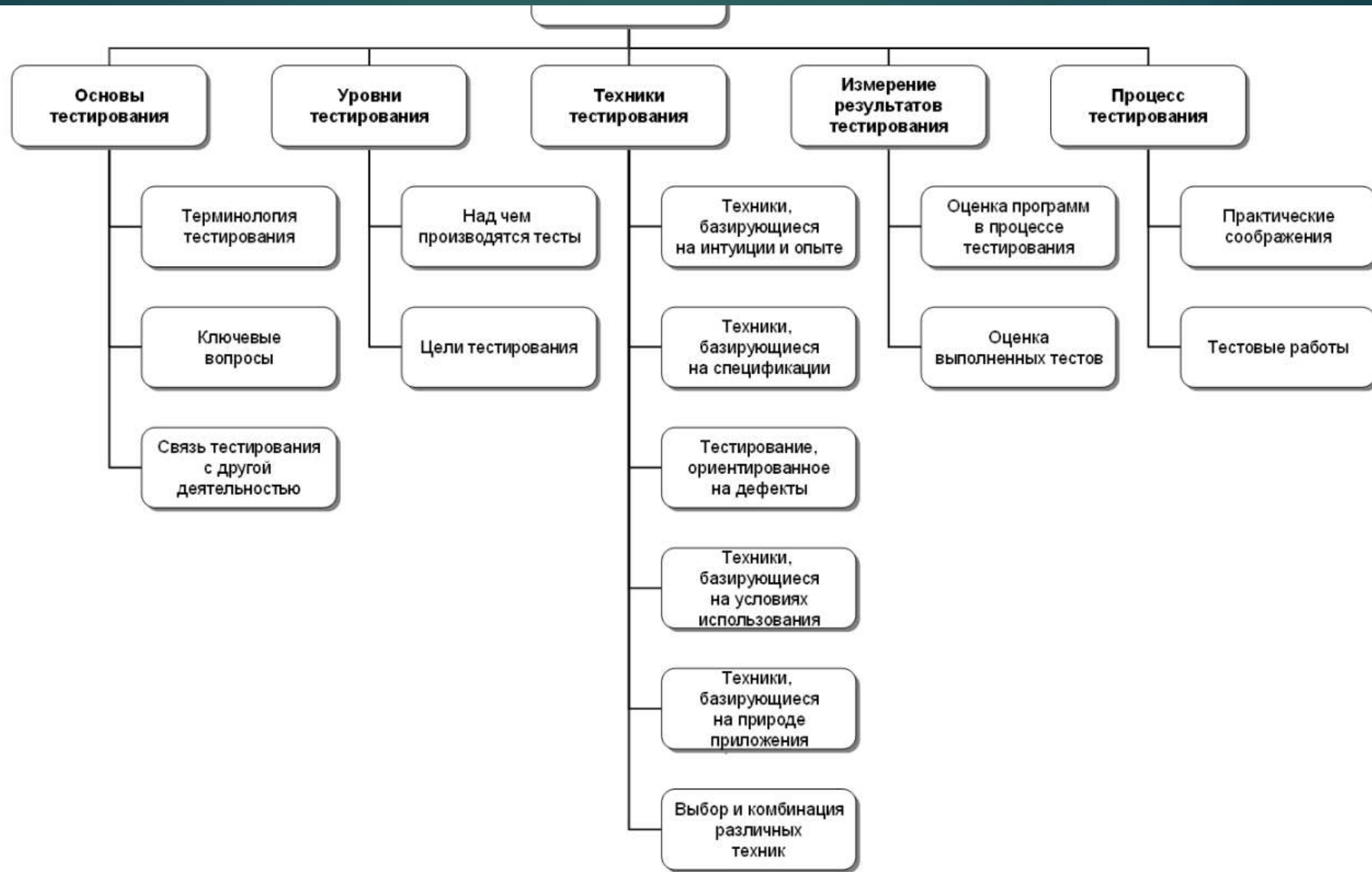
class CalcBmi implements ICalcBmi {
    public float calculate(float h, float w) {
        return w/h/h*10000; }
}

class CalcIhw implements ICalcBmi {
    public float calculate(float h, float w) {
        return (h-w)/100; }
}
```

- ▶ Интерфейс модуля
- ▶ Внутренние структуры данных
- ▶ Независимые пути
- ▶ Пути обработки ошибок
- ▶ Граничные условия

Моки и стабы





Тестирование – сродни искусству

ИЛИ

<https://habrahabr.ru/company/sqalab/blog/217743/>



До тестирования



После тестирования

Не бывает совершенных программ.
Бывают неотестированные.